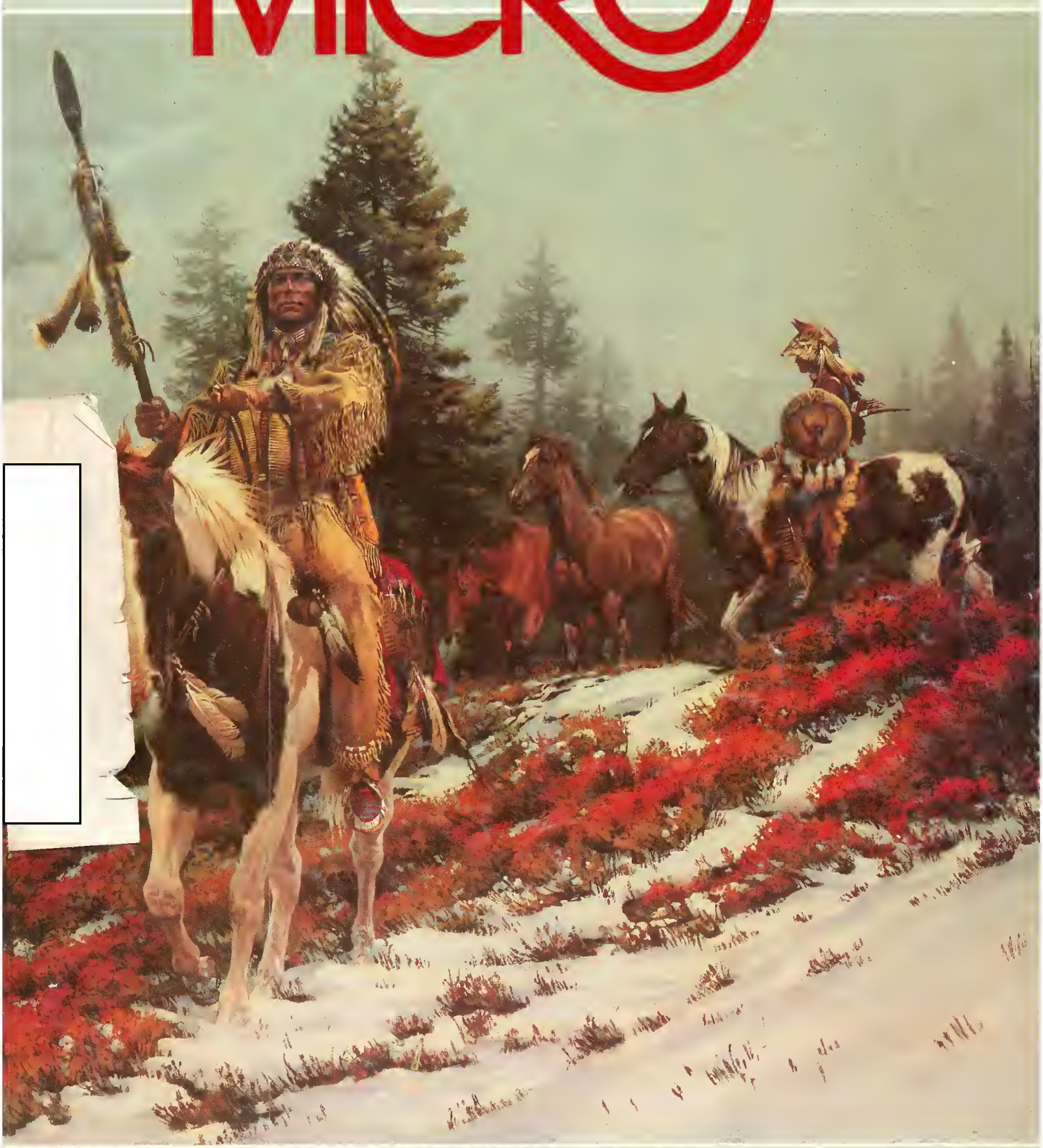
IEEE MICRO

AUGUST 1988



THE COMPUTER SOCIETY

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

Fault tolerance:
Standby spares ensure the mission

Cover image: "Windrider" by Chuck Ren
Cover design: Jay Simpson

IEEE MICRO

Volume 8 Number 4 (ISSN 0272-1732) August 1988

Published by the Computer Society

Departments

Feature Articles

Special Features

IEEE Micro

The Computer Society
10662 Los Vaqueros Circle
Los Alamitos, CA 90720
(714) 821-8380

*Submit six copies of all articles and special-issue proposals to James J. Farrell III, 8375 South River Parkway, Tempe, AZ 85284; (602) 752-6222; Compmail+, j.farrell.

Moving?

Address changes: Please notify us 4 weeks in advance

Name (Please Print)

New Address

City

State/Country Zip

ATTACH LABEL HERE

- This address change notice will apply to all IEEE publications to which you subscribe.
- List new address above.
- If you have a question about your subscription, place label here and clip this form to your letter.

Mail to:
IEEE Micro, Circulation Dept., 10662 Los Vaqueros Cir., Los Alamitos, CA 90720-2578

Circulation: *IEEE Micro* (ISSN 0272-1732) is published bimonthly by the Computer Society of the IEEE: IEEE Headquarters, 345 East 47th St., New York, NY 10017; Computer Society West Coast Office, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578. Annual subscription: $18 in addition to Computer Society or any other IEEE society member dues; $27 for members of other technical organizations. Single copies: $10 for members; $20 for nonmembers. This journal is also available in microfiche form.

Postmaster: Send address changes and undelivered copies to *IEEE Micro*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578. Second class postage is paid at New York, NY, and at additional mailing offices.

Editorial: Unless otherwise stated, bylined articles and descriptions of products and services in New Products, Product Summary, MicroReview, MicroNews, MicroView, and MicroLaw, reflect the author's or firm's opinion; inclusion in this publication does not necessarily constitute endorsement by the IEEE or the Computer Society of the IEEE. Send editorial correspondence to *IEEE Micro*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578. All submissions are subject to editing for style, clarity, and space considerations. *IEEE Micro* subscribes to The Computer Press Association's code of professional ethics. VBPA

Department

From the Editor-in-Chief

Meetings in Anaheim

During the mid-June Design Automation Conference in Anaheim, California, the Computer Society held its usual periodic meetings of the various and sundry boards and committees that make the society function. This event usually occurs three or four times a year. In addition, the week marked the annual gatherings of the Magazine Planning Workshop for society magazine editors and professional staff, and the Editorial Board meeting for *IEEE Micro*. Like all Computer Society members, I am welcome to attend any meeting of my interest. Because of my schedule and other commitments, I must limit my attendance to those boards and committees of which I am a member (Magazine Advisory Committee, Publications Board, Magazine Workshop, *IEEE Micro* Editorial Board). All in all, it was a busy week.

In no particular order follow the events and results of these meetings.

Magazine Workshop

We examined desktop publishing, which was explained rather thoroughly. A good deal of work has already been done to implement desktop publishing in Computer Society publications. Desktop publishing greatly reduces fixed costs per editorial page, thereby containing costs for Computer Society members.

The Computer Society presented Marie English, our managing editor, with a Meritorious Service Certificate for her performance on *IEEE Micro*. She certainly deserves this award. Her excellent efforts are evident to all who read this magazine.

Magazine Advisory Committee

Since my final term as editor-in-chief expires with the December issue of this year, a search committee has been formed and chartered to submit the name of an EIC candidate for *IEEE Micro* at the November 1988 Computer Society meetings.

Publications Board

On a year-to-year basis, *IEEE Micro* is increasing its circulation. In addition, advertiser interest appears to be growing. This is good news.

IEEE Micro subscription rates for 1989 have been requested at $18 for members, $33 for members of sister societies (like ACM), and $129 for nonmembers (libraries).

IEEE Micro Editorial Board

The board agreed to expand the number of European editors by two: one to assist Dante Del Corso and Karl Grosspietsch with article work, and one to initiate the European aspect of a new technology column to be named MicroWorld. I am also soliciting a new Far East editor to handle that geographical aspect of technology for MicroWorld.

In other actions, the board selected "The TRON Project" by Ken Sakamura (April 1987) as the best article of 1987 appearing in *IEEE Micro*. We submitted additional articles for consideration of other IEEE publication awards.

The board also updated the 1989 Editorial Calendar and generated the 1990 Editorial Calendar. The new editor-in-chief will make the guest editor assignments for 1990 in January 1989. In addition, the Editorial Board suggested the following topics for solicited manuscripts: multi-microprocessing, operating systems, bioengineering, personal computers, coprocessors, ASICs, and device packaging.

Marie English is producing a new "Information for Authors" guideline for 1989. While it is important to maintain our level of technical depth, we plan to restrict articles to a maximum of 15 magazine pages. With occasional exceptions, we feel we can maintain technical depth in our contributed articles in less than 15 pages.

Dave Gustavson is generating a new reviewer/referee form. Article length will be a reviewing consideration.

Christine Miller, our assistant editor, suggested a regular short tutorial column that is contributed by a well-known authority in his or her field. We will try this experimentally during 1989. I am seeking a guest editor to head up this effort.

Marie English will continue with a short "newsy" box, renamed MicroBits, that contains special-interest items capsulized especially for our readers.

The Editorial Board decided to hold its next meeting in Anaheim again. The new EIC will publish the date and details in January.

Jim Farrell

Letters to the Editor

The mailbag:

"I liked the overall view provided on the Japanese microprocessor situation. Very good issue." A.C., Sherbrooke, Canada (I have had several similar verbal comments as well. We intend to maintain the April issue focus on the Far East, expanding coverage to the "Four Tigers" when technology dictates.—J.F.)

"I liked all articles (especially GaAs R&D)...more on Neural Networks (tutorial) and LANs." N.M., Split, Yugoslavia (I am sure that Milutinovic appreciates your comment. We are planning more tutorial features.—J.F.)

"I liked everything!...more on Occam and Transputers." (Dec. 1987). "I liked MicroView very much! ...and, of course, The mailbag." (Feb. 1988). H.R.B., Palomar, Argentina (I am glad to see your input. I was afraid that the transputer was getting overexposed. I have just received a new article on the transputer from Switzerland. It is now in review.—J.F.)

"I would like to see more on new product reviews, benchmark comparisons, and professional comparative evaluations." R.P.N., Muscat, Oman (Several US trade publications go into great detail with new products. Since this information is not generally available elsewhere—where *Micro* is—*Micro* tries to focus on products of direct interest to our readers, with reasonable depth. Large numbers of readers like benchmarks, but getting them from a completely objective source can be difficult.—J.F.)

"I would like to see Novix Forth applications." D.N.M., Haifa, Israel

"I liked the RISC article...more advertisements." R.C.S., Fayetteville, AR

"I liked New Products." I.N., Kanata, Ontario, Canada

"I was particularly impressed to see the article on the μBTRON and how it worked....' M.S., Oakland, CA

"I liked MicroLaw...extremely important to me. (liked) MicroNews Forth Inc., benchmarks...would like to see reviews of buses, software technology, and periodicals." A.F., Oxnard, CA

"I liked the selection of articles." A.E., Sirintepe, Turkey

Continued on p. 85

A Good Idea Has Arrived

To the Editor:

I am delighted that MicroLaw Editor Richard Stern has invited me to continue this timely discussion on the Berne Convention begun in the April issue.

On May 10th the full House of Representatives passed H.R. 4262, the Berne Convention Implementation Act of 1988, by a vote of 420 to 0. A Senate vote passing S. 1301, the counterpart legislation, is expected before the publication date of this article. If the United States now adheres, after an absence of more than 100 years, it will finally provide its authors the same foreign copyright protection that almost 80 nations now adhering to the 1886 Berne Convention provide their authors. Up to now, the only major nations that have not been Berne members have been the Soviet Union and the People's Republic of China—and the United States.

Literary piracy goes back a long way; there were probably clay-tablet plagiarists 5,000 years ago in Sumer, but recent advances in copying technology have made modern piracy possible. The 15th century introduction in Europe of movable type facilitated relatively inexpensive reproduction of literary works. For the first time western authors and artists saw others readily exploit their creativity. The German illustrator Albrecht Durer at the time successfully prevailed on municipal leaders to stop unauthorized copying of some of his book engravings. During the 19th century, many books by the American author Samuel Clemens (Mark Twain) were printed outside the US, without permission or payment, in inferior editions which often appeared in the US in competition with the authorized versions that provided his livelihood. As we approach the 21st century, the pace of piracy has quickened enormously.

Last April Dennis Karjala found "almost surely overstated" my statement that US authors and copyright proprietors lose billions of dollars from foreign piracy. But a US International Trade Commission report published this February estimates worldwide losses to US owners of intellectual property due to inadequate protection at a staggering $23 billion in 1986. Given the mind-boggling size of the losses to foreign pirates, it is inexplicable that we have waited so long to give US authors effective means to catch the pirates in their safe harbors. Administration officials most familiar with the problem of piracy, including Secretary of Commerce Verity (and his predecessor, Malcolm Baldrige), US Trade Representative Clayton Yeutter, and Under Secretary of State Allen Wallis, have been unanimous in their view that Berne adherence would help significantly in the fight against international copyright piracy.

Berne will give the US copyright relations with 24 additional countries. These countries are members of Berne, but they don't belong to the Universal Copyright Convention (a treaty the US adheres to that is less effective than Berne), and the US has no effective bilateral relations with them. They in-

Continued on p. 86

Department

MicroLaw

Richard H. Stern
Law Offices of Richard H. Stern
2101 L Street NW, Suite 800
Washington, DC 20037

Copyright Office rejects Computer Society position

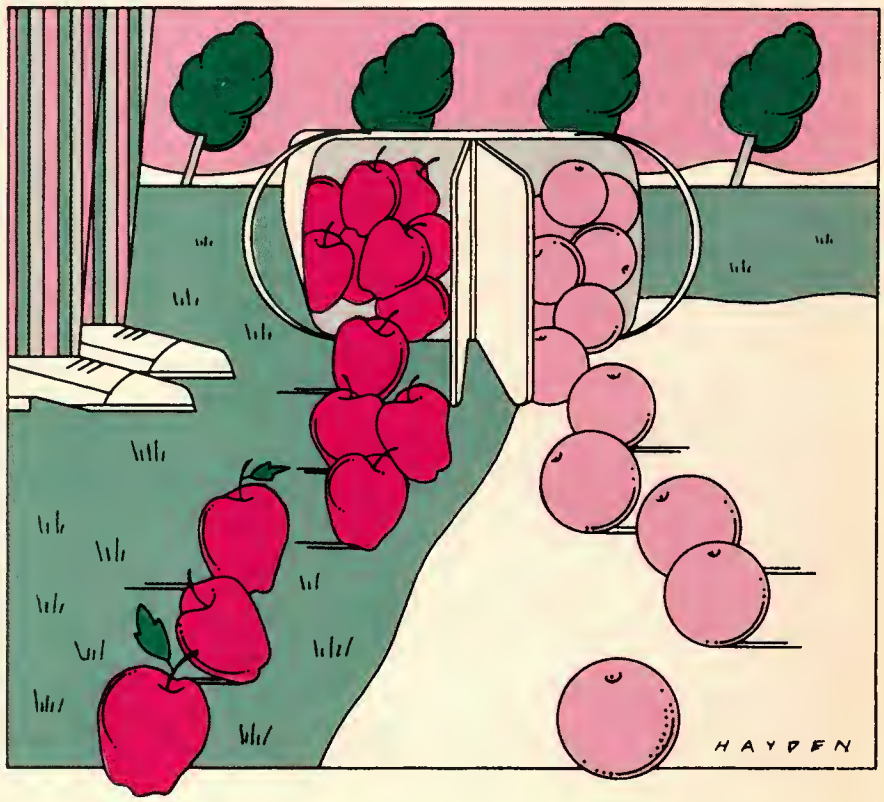
In the fall of 1987, the Copyright Office invited technical input on proposed rule making for copyright registration of screen displays. In response to this invitation, the Computer Society (acting through its Committee on Public Policy, or COPP) submitted a position paper. This paper discussed whether screen displays should be considered as part of their underlying computer programs—or as separate works of authorship. It also examined the likely consequences for software economics and progress should screen displays and underlying computer programs be registered separately—or as a unit—for copyright purposes. COPP/Computer Society representatives (including this writer) then explained the paper at a Copyright Office hearing on the issue. The Board of Governors of the Society also adopted and forwarded to the Copyright Office a resolution calling for *separate registration* of computer programs and their associated screen displays.

The Copyright Office is fanatical about "unitary registration of copyright."

The Computer Society position is that authorship of computer programs and screen displays differs in several ways. Many programs can cause the same screen display, and slight variations in the code of a computer program can greatly change the screen display it generates. Also, different persons typically prepare the screens and code. The processes of creating screens and code are quite different, and what should be protected in a well-ordered universe is different with regard to those two things. In particular, screen displays raise problems about appropriation or monopolization of the teachings of the science of human factors analysis and standardization of user interfaces. That is, should users have legal rights in regard to *habit*, *cost*, and *unwillingness* when it comes to the burden of having to learn how to use a new interface?[1,2]

After considering the matter for nine months, the Copyright Office published a disappointing and terse notice (*Notice of Registration Decision*, June 3, 1988). It ruled that it will require screen displays and the computer programs with which they are associated to be registered as a *single work*, at least whenever the two things are owned by a single person or business entity. Time and space constraints prevent a detailed analysis of the Copyright Office action here—that will have to wait a few months. However, the basic outline of the Copyright Office's position follows.

The Copyright Office is fanatical about a principle it calls the "unitary registration of copyright." It wants one copyright registration for one work, lest more than one registration for



a work tear apart the fabric of intellectual property law. The mechanism of the catastrophe is unclear, but the Copyright Office wants to take no chances.[3] Accordingly, it resolves any doubts in favor of fewer rather than more copyrights for any item, including software.

Under this analysis—if that is the right word—it is immaterial that different standards of copyrightability or different scopes of protection are appropriate for different things crammed into the same portmanteau. It is also immaterial that it is easier to make distinctions of the types just suggested when the different things are (at least potentially) separately registered, licensed, and enforced.

The Copyright Office has no interest in addressing the problems involved in both avoiding patents on the teachings of human factors analysis, and at the same time giving adequate encouragement to capital formation in the software industry. It apparently hopes that the question will go away if ignored. The Office stated that it "is sympathetic to users who may have difficulty in determining the scope of copyright in computer software," but the Copyright Office cannot deal with that problem and will have to leave it to the courts. At most, the Office feels that it can assist the public and the courts by refusing to register any clearly "frivolous, unsound, or otherwise unjustified claims."

The Copyright Office *will* permit applicants to deposit printouts, photographs, drawings, or videotapes of screen displays. It will also permit applicants to mention in their applications that they claim protection for the screen displays as well as the code of the computer program they are registering.[4]

Curiously, the Copyright Office parenthetically mentioned the *Softklone* decision of the Atlanta federal district court. It acknowledged that *Softklone* required that a separate claim to, and registration of, screen displays (as distinguished from the associated code) must be made for copyright protection. But the Office did not then discuss why the *Softklone* decision was wrong or why it refused to follow *Softklone*. The Copyright Office did say, however, that it is convinced that its single registration policy is sound and "the Office assumes that this decision makes clear to the public and the courts our view that multiple claims are unnecessary, and indeed not appropriate, to assert copyright in screen displays."

Where does this leave us? Option one is to acquiesce in the fiat of the Copyright Office and file a single application for registration of the computer program code and screen displays, depositing copies of both. Those who follow this policy have to cross their fingers about getting into litigation over the screen displays in the Atlanta federal district court or any other court that believes that the Atlanta court correctly decided the *Softklone* case. Such litigants would have to argue that the Copyright Office's decision on registration makes more sense than the court's decision in *Softklone*, or else supersedes it because of the superior authority of the Copyright Office in the field of interpreting the Copyright Act.

The purest approach requires industrial-grade dedication to quixotism.

Some defer to the interpretation that an agency gives to the statute under which it operates. The very recent *K Mart/Copiat* decision of the Supreme Court concerned this point. The issue was the validity of Customs Service regulations that refused to bar parallel importation of genuine trademarked goods from a foreign company affiliated with the same US company that seeks to prevent the parallel importation ("grey marketing"). This situation can occur when the two companies are parent and subsidiary. Two Justices opined that the regulations should be upheld unless the Customs Service's interpretation of the statute was clearly wrong, which they felt was not the case. Three Justices opined that the Customs Service's interpretation was sound and should therefore be upheld. They considered this conclusion to be bolstered by the deference due the agency's interpretation of its statute. Four Justices dissented because they disagreed with the Customs Service's interpretation of the statute and did not feel compelled to give it much deference.

This all suggests that option one presents somewhat of a horse race.

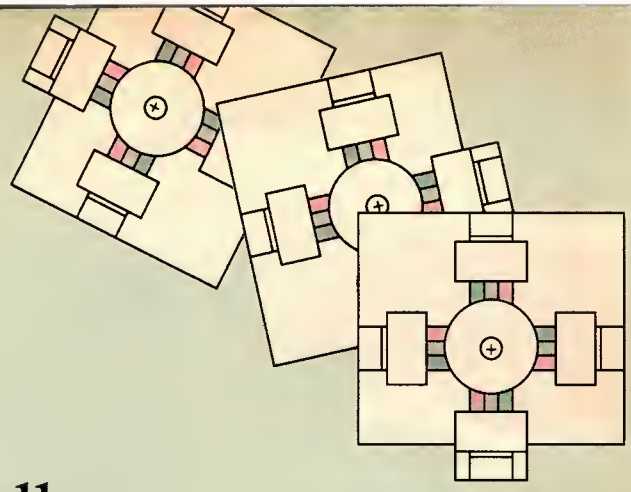
Option two is to present two applications to the Copyright Office (one for the code and one for the screen displays), let the Copyright Office refuse to register both, and then go to court with the Copyright Office. That action would preferably be taken by a software company located in Atlanta so that it could sue the Copyright Office before the same federal court that held in *Softklone* that the two things should be separately registered. The idea is to vindicate the position of the Computer Society and those others who believe that screen displays and code ought to be treated differently. This approach is the purest, but it requires industrial-grade dedication to quixotism.

A third option—less pure but cheaper—is now receiving my serious personal consideration. Option three is to assign the copyrights in the code and screen displays to separate companies (for example, Jones Code Corp. and Jones Screen Display, Inc.). Since the Copyright Office is insisting on unitary registration apparently only where the same company owns both code and screen displays, suitable legal mumbo jumbo can avoid the issue.

References

1. R.H. Stern, MicroNews, *IEEE Micro*, Vol. 7, No. 6, Dec. 1987, p. 90.
2. R.J. Preiss, Computer Society News, *Computer*, Vol. 20, No. 10, Oct. 1987, pp. 75-76.
3. The Copyright Office said: "Permitting multiple registrations of parts of works would increase confusion among those attempting to use the records of the Copyright Office. Subdividing claims might also result in multiple infringement actions and multiple claims for statutory damages, based on separate registration. The Copyright Office benefits by having a simplified administrative process." Presumably, only the last point really counts. The Copyright Office does not explain how confusion would be increased by separating claims to code and screen displays, and the assertion is not inherently plausible. The risk of multiple actions is slight or even farfetched. A later comer ordinarily emulates only the user interface and does not attempt to copy the code—except in cases of the most blatant piracy. These cases are not only uncommon, but also inappropriate. Who cares when such a defendant gets socked with multiple infringement claims? In any event, statutory damages are discretionary with the court, so that abuse as to their amount is unlikely.
4. The suggested language is "Nature of authorship: Computer program code and screen displays."

FEATURE



A Fault-Tolerant Multiprocessor Controller for Magnetic Bearings

When reliability is critical, try adapting this control system to your particular problem. It translates well to a variety of applications.

Steven W. Yates
Ronald D. Williams

University of Virginia

Multiprocessors offer many desirable properties such as significant system throughput and the potential for fault-tolerant performance. The incorporation of fault tolerance in multiprocessors is typically accomplished through the inclusion of fault detection and location routines and redundant processing modules. As advances in VLSI component technology allow processor performance to increase with corresponding reductions in size and power consumption, multiprocessor systems will be more widely used.

Unfortunately, we do not easily attain the benefits associated with multiprocessor architectures because the partitioning of general computing tasks can be very difficult.[1] In general, extraction of the parallel components of most algorithms is very difficult. Also, once an algorithm has been converted to a parallel form, optimal assignment of processing elements is also difficult. Finally, fault-tolerant systems that employ software-intensive fault detection and location can spend a significant amount of processing time executing self-diagnostic routines. In real-time systems this overhead may reduce system throughput to an unacceptable level.

Here, we describe a fault-tolerant multiprocessor system designed to provide digital control for an active magnetic bearing. This particular control problem offers an interesting application for fault-tolerant multiprocessing because the problem demands significant processing power, is easily divided into relatively independent parallel tasks, and cannot tolerate controller failures. Even though we designed this system for a specific problem, we expended much effort to make the system design general enough for successful application to a wide range of real-time applications.

Active magnetic bearings are beginning to be used instead of conventional roller ball or fluid film bearings. They are used in applications that require low friction, long bearing life, and modification of the dynamic characteristics of rotating machinery. Figure 1 shows a typical magnetic bearing test rig designed for laboratory experimentation. In this configuration, the machine shaft is centered vertically and horizontally at both ends by forces generated by the bearing electromagnets. This type of open-loop unstable system needs active feedback to stabilize the rotor. The typical rotor supported by magnetic bearings has five axes of control: x and y axes for each of the two radial bearings plus a z axis for a thrust bearing. Each of the five axes exhibits a high degree of autonomy, but coupling between them is also possible. In our current laboratory setup, we do not need a thrust bearing because of the axial restraint provided by the motor and flexible coupling.

0272-1732/88/0800-0006$01.00 © 1988 IEEE

While simple analog controls stabilize most existing commercial magnetic bearings, the presence of active control offers the potential for significant improvement in the dynamic behavior of the machine. An active magnetic bearing with simple analog controls offers many advantages over conventional bearings, so it seems reasonable to increase the controller complexity to attain other desirable design goals.

For example, adaptive control algorithms could enable a high-speed turbomachine to tolerate the loss of a turbine blade. Also, rotational speed-dependent algorithms allow the bearings to exert increased position stiffness and damping on the rotor near its critical speeds, while reduction of the stiffness and damping at other speeds reduces bearing power consumption. These and other complex control algorithms are difficult if not impossible to implement using analog electronics, so they provide an opportunity for the effective use of a multiprocessor system.

Literature reports limited use of digital controllers for magnetic bearings and other magnetic suspension systems. However, none of these applications have made use of a fault-tolerant digital controller.[2,3] In addition, no use of an architecture optimized for the magnetic bearing-control problem has yet been reported. We describe such a system.

Overall system requirements

Since we focus on the development of a multiprocessor system for the control of magnetic bearings, the magnetic bearing-control problem constitutes an important basis for the system specifications. The magnetic bearing places some stringent requirements upon system throughput and reliability. To control high-order vibration modes in high-speed rotors, we require sampling rates in excess of 100 kHz per axis of control. Although lower sampling rates may be used to stabilize the rotor, effectively controlling these high-frequency vibration modes requires sampling rates of this magnitude. Also, the sampling rate should not decrease significantly as adaptive or other complex control algorithms are used.

The system's mean time to failure, or MTTF, must be at least 40,000 hours for the digital controller to exhibit reliability superior to that of an analog controller. We considered this figure to be an absolute minimum design goal for this project, although increases in MTTF beyond this level are certainly desirable. To decrease the probability of design errors entering the system, we used tested and proven existing commercial single-board computers and bus standards wherever possible. Likewise, to reduce the probabilities of software errors occurring and to reduce the debugging time for those errors that do occur, we wrote the system code in a high-level language where possible.

While the use of standard components is generally expected to reduce the system development time significantly, the use of standard single-board computers and bus products necessitates software-intensive techniques for fault detection and location. This fault detection and location must also include all components in the signal path, both analog and digital. Therefore, the system must provide the capability of performing functional tests of the analog-to-digital (A/D) and digital-to-analog (D/A) converters as well as any other analog components, such as analog multiplexers and filters.

The intended application offers considerable freedom in the implementation of processor reconfiguration in the event of a failure. Although the system should normally control high-frequency shaft vibrations at the bearings, it is possible to provide incorrect outputs to the bearing for relatively long periods of time. It takes several tens of

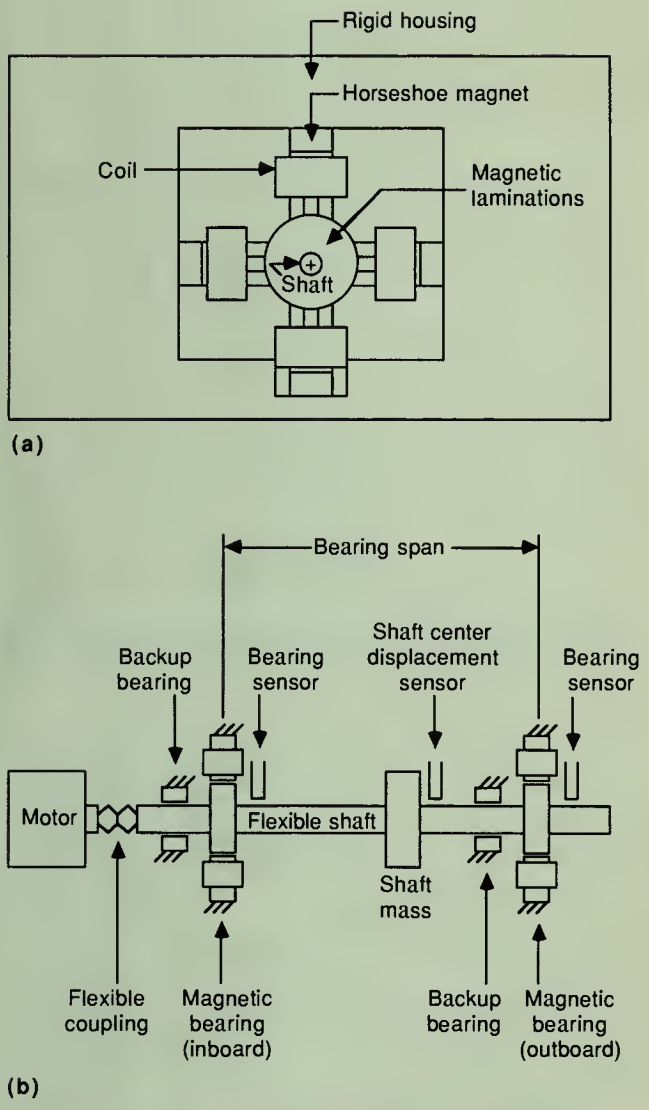


Figure 1. Magnetic bearing end view (a); typical magnetic bearing test rig (b).

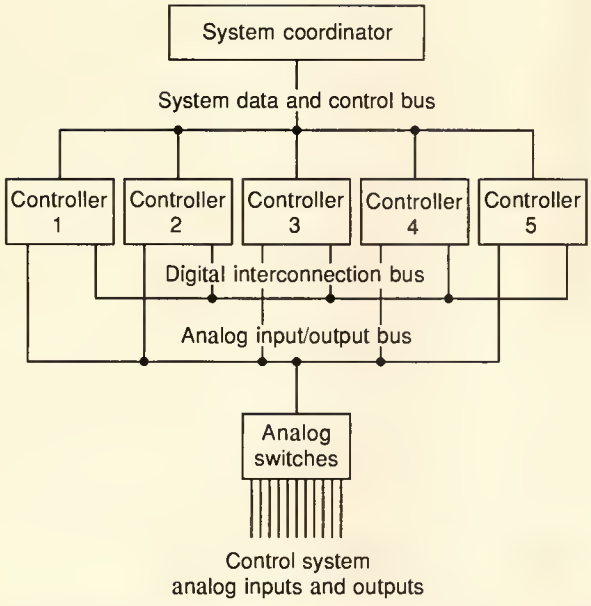


Figure 2. System structure.

milliseconds for the rotor to contact the mechanical backup bearings after control has been lost. Therefore, we found it useful to provide a precise definition of the term "fault tolerance" in the context of this application. We consider this system to be fault tolerant if a partial computer system failure produces no significant deviation of the rotor from its desired position.

Our architecture may skip one sampling period during dynamic reconfiguration of processors. However, due to the relatively long time required for the rotor to fall out of support when the outputs are not updated, these skipped outputs will not cause any significant perturbation of the rotor. We further reduced the effects of these types of transient errors by using an analog input/output (I/O) board. Since sample-and-hold devices store the analog outputs, an output only one sampling period late will not differ enough from the correct output value to cause catastrophic shaft-behavior deviation.

Since the system is intended for experimentation, we found ease of use to be essential. Also, we wanted to be able to increase system reliability and throughput by adding redundant processors and by making any desirable software modifications. Many other possible applications of this architecture may not require the high level of speed and reliability required by the magnetic bearing. So, the architecture must allow operation with fewer processors at a reduced level of system performance and reliability. This capability also allows the system to recover from faults with a graceful degradation in processing capabilities. Finally, we designed this architecture to permit simple incorporation of newer, faster processors in both levels of the system as they become available.

System architecture

To provide the capabilities for high sampling rates, high reliability, easy system expansion, and good overall system flexibility, we chose a two-level hierarchical structure for the system architecture, as illustrated in Figure 2. This hierarchy of computers permits powerful digital signal processing, or DSP, modules to process sample-to-sample control algorithms rapidly for each bearing axis. At the same time, the general-purpose supervisor computer performs higher level coordination and adaptation. This multiple computer system permits the natural parallelism of the magnetic bearing problem to be exploited. It also provides graceful degradation of performance in the event of a partial computer system failure. The structure includes much processing power in the lower level, direct digital control modules. The processor in each of these modules is an optimized digital signal processor. This architecture provides several significant advantages in a distributed control application:

- Substantial computing power in the direct digital control modules can be reassigned to provide emergency control and graceful degradation in the event of the failure of one of the modules.
- The modular structure of the controller simplifies software development. Much of the software is redundant among modules.
- This arrangement offers significant fault isolation since each direct digital control module controls only a small, relatively autonomous part of the total physical system.

Our architecture is similar to the Software-Implemented Fault Tolerance, or SIFT, computer developed by Wensley et al. at the Stanford Research Institute for the National Aeronautics and Space Administration.[4] This architecture was previously applied to an industrial controller by August Systems in the Series 300 controller.[5]

One similarity in the architectures occurs in the low-level control structure. Here, a loosely coupled, loosely synchronized multiprocessor system contains sufficient excess computational capabilities to allow processors to be taken off line for testing periodically without reducing the system functionality. In the SIFT architecture, three processors normally support each task, with all three performing software voting on the inputs and outputs of each task. Voting is not performed on intermediate calculations. The architecture we describe supports such voting if it is needed to achieve a given reliability requirement. However, voting does not appear to be needed to achieve the desired reliability.

Another similarity to the SIFT architecture is the use of loose synchronization among the low-level processors. Instead of using a centralized clock generator for all of the low-level processors and requiring them to operate in perfect synchronization, each processor con-

tains its own independent clock generator and may be as much as 100 nanoseconds out of synchrony with any other processor.

Use of this approach to synchronization has several important advantages in the control of magnetic bearings. For example, the effect of a clock generator failure becomes localized to only one processing element. Clearly, a failure in the central system clock cycle would have devastating results for the magnetic bearing. Although fault-tolerant clock generators have been built successfully, they usually require the use of rather sophisticated phase-lock techniques. These techniques are not very attractive in our application because of cost constraints. Further, no artificial synchronization requirements are placed on the individual controllers. They may even execute entirely different control algorithms, if desired. The only constraint is that the processor must complete the computation of an output value within the preset sampling period (currently 10 microseconds).

Our system is fault tolerant if the rotor doesn't change position upon partial system failure.

Digital Control Module, or DCM, boards, which are based on the Texas Instruments TMS320C25 DSP chip, provide the lowest level, direct digital control of the magnetic bearing. Because of limited availability of this chip during the design of the DCMs, we substituted a pin-compatible TMS32020 in the prototype design. The major difference between these chips is the maximum clock rate: the TMS32020 can operate at a maximum clock frequency of 20 MHz, as compared to 40 MHz for the TMS320C25. We designed all of the circuitry on the DCM so that a faster TMS320C25 and a 40-MHz oscillator could be installed without hardware modification. The chip change would double the system throughput.

Each DCM contains its own high-speed A/D and D/A converters as well as analog input and output buffers and multiplexers. In addition, each DCM includes hardware to communicate its current inputs and outputs to all other DCMs in the system with no software overhead. In normal operation, the system would contain five of these boards, with each assigned to control one bearing axis. Any DCM can control any bearing axis or can be deactivated by the system supervisor. In addition, each DCM can be multiplexed to control more than one axis with a corresponding drop in the sampling rate.

The system supervisor is a commercially available Motorola 68020-based, VMEbus CPU with a 68881 floating-point coprocessor operating at a 16.67-MHz clock rate. This module manages the software-implemented fault-tolerance capabilities of the system, implements adaptive control algorithms, and interfaces with the user and host computer system. All of the supervisory software is written in C. Although the current design includes only one supervisor, another could be added to increase the reliability and throughput at this level of the hierarchy.

Since any DCM may be requested to control any axis at any time, each DCM must have the result of every other DCM's recent A/D conversions and control outputs. This loose coupling of processors also allows straightforward implementation of multiple-input, multiple-output state control algorithms. The entire low-level portion of the system is thus a loosely coupled multiprocessor structure. Each processor contains local program and data memories that are shared by the VMEbus, analog input/output, and intercommunication hardware. Each knows the other's current inputs and outputs with no software overhead.

We used a novel approach in the design of the low-level intercommunication network. This network consists of five independent, high-speed serial buses, each with its own clock and data signals. The novel aspect of this digital intercommunication scheme is its implementation. At the start of every sampling period, each controller initiates an A/D conversion cycle. Analog Devices HAS-1204BM 12-bit converters offer a 2-μs total sample acquisition and conversion time.

As conversion takes place, each A/D converter broadcasts serial data and clock outputs over a dedicated serial bus. Each DCM has five 16-bit shift registers to receive the data on each of these intercommunication channels. By using this technique, every DCM receives the A/D conversion result of every other DCM in parallel with the actual conversion process with absolutely no software overhead and very modest hardware requirements. Thus, every processor receives the conversion results available as soon as each conversion cycle completes. In this application, the use of a serial intercommunication scheme imposes no loss of system performance. In fact, we found no faster method of distributing all of the control information among processors.

Partitioning the system into a two-level hierarchy also allows distinct fault detection and isolation techniques to be used at each level. At the lowest level, each DCM has a programmable watchdog timer. The period of this timer is set in accordance with the sampling period associated with the current control algorithm by the system supervisor. A watchdog time-out interrupts the system supervisor, which then determines which DCM was at fault. Each DCM can also perform a self-diagnostic test sequence at the request of the system supervisor to perform checks of its processor, bus interface, memory system, and analog components. However, due to the relatively stringent throughput requirements imposed on the DCMs by the magnetic bearing-control problem, we incorporated only a small amount of software redundancy in the actual digital control code.

At the highest level, the system supervisor monitors the health of the individual controller boards to ensure that the expected control actions are being performed. In the event of a failure of any system component, the supervisor minimizes the impact on system performance by reconfiguring the remaining processors among the control channels. The failure of a supervisor yields a loss of the high-level functions, but the low-level control structure continues to operate normally. However, if one of the low-level digital control processors subsequently fails, reconfiguration is not possible, and a catastrophic system failure results. To prevent the loss of high-level supervisory functions, users can add redundant system supervisors. The failure of any module causes an alarm to be sounded so that repairs can be made to restore the system to its fully operational state. However, the loss of a module will not drop the spinning rotor from support.

Because the individual DCM boards can be assigned to control any bearing axis, we needed some means of storing the analog outputs on each analog bus. We assigned a dedicated bus to each axis of bearing control, in addition to an analog channel dedicated strictly to testing the analog I/O module. The DCMs multiplex their own inputs and outputs from these buses after receiving their channel assignments from the system supervisor.

During system initialization, the supervisor must not assign more than one DCM to each control axis. Obviously, if two or more DCMs are trying to output to the same analog bus at the same time, contention problems result. An analog I/O board, which contains analog input and output filters, differential line drivers, and sample-and-hold devices for each control channel, stores the analog output information. This analog switch board can be included in the loop during analog component tests, so that the supervisor can identify faults on it or anywhere else in the analog signal path. Redundant analog switches allow reconfiguration around failed components. The supervisor controls these analog I/O modules by either enabling or disabling them.

DCM design

A block diagram of the DCM architecture appears in Figure 3. Note that each DCM includes 8 kilobytes of zero-wait-state, 30-nanosecond program and data RAM. The TMS320C25, based on a modified Harvard architecture, includes separate program and data memory spaces. Internally, the TMS320C25 uses parallel data and program buses. But the external interface uses only one 16-bit bus shared by the program and data memory spaces and the I/O devices.

Each DCM also includes 8 Kbytes of erasable, programmable ROM (EPROM) for software storage to facilitate communication with the supervisor as well as any low-level diagnostic routines. To ensure that the prototype DCM works properly at the 40-MHz clock cycle rate of the TMS320C25, we paid careful attention to circuit layout and wiring practices. All wiring appears on a multilayer VMEbus prototype card.

Redundant analog switches allow reconfiguration around failed components.

To allow efficient communications between each DCM and the supervisor, we included a sophisticated VMEbus interface circuit. Each DCM board maps into a 256-Kbyte portion of the VMEbus global address space, and all of the memory and I/O devices on each DCM becomes available to the VMEbus. The HOLD´ logic internal to the TMS320C25 arbitrates between the local DSP chip and the VMEbus. Each DCM contains logic to monitor the VMEbus address signals and address modifier codes. If both a valid board address and a valid address modifier code are received, the system generates a HOLD´ request and presents it to the TMS320C25. After the execution of its current instruction, the TMS320C25 relinquishes control of its local bus and asserts HLDA´ to indicate that it has placed its buses in a high-impedance state. This acknowledgment enables on-board tristate buffers that place TMS320C25 format control signals generated from the VMEbus control signals on the local DCM control lines. Thus, the VMEbus and the TMS320C25 appear as identical bus masters to the memory and I/O devices on each DCM.

We achieve several significant advantages with this approach. For example, all of the memory decoding and READY signal-generation logic used by the TMS320C25 can also be used by the VMEbus interface. This is a very important point, because the VMEbus requires an acknowledge signal for all data transfers (DTACK´). Sharing this logic eliminates the need for a duplicate VMEbus address decoder and DTACK´ generator. Also, use of the existing READY signal generator helps to increase the reliability of the system. If the system supervisor initiates a memory cycle using an unassigned address, and even if the address is within the valid 256-Kbyte range of one of the DCMs, it receives no acknowledgment, and the supervisor's bus timer generates a bus-error exception to call the supervisor's attention to the problem.

Also, the fault detection and location routines executed by the supervisor become more effective when direct access is possible to every device on every DCM. This effectiveness results from the supervisor's increased ability to observe the overall system. Note that since each DSP chip normally executes out of its internal data

and program RAM to increase throughput, the DSP chips do not need to use their external buses except when inputting and outputting. Therefore, this method of intercommunication imposes only a very small overhead on the DCM because of the relatively small amount of input and output the DSP modules actively perform.

The system supervisor passes messages to the lower level DCMs by writing to their memories. In normal operation each low-level processor executes out of its on-chip data and program memories. When the supervisor wishes either to change some of the control parameters or to load an entirely new control algorithm, it simply writes the data directly to the proper DCM's external RAM. Each DCM contains an 8-bit control/status register that determines which channel the DCM is controlling, activates and deactivates the DCM, and tells the DCM when it has been given new data and instructions by the supervisor. After the new data has been written, the supervisor sets the DATA and INSTR bits of the control/status register and interrupts the DCM. By examining the DATA and INSTR bits of the control/status register, the DCM determines what type of message was passed to it, loads the new parameters or algorithms into its on-chip memory, and continues execution.

Interrupts perform several functions within both levels of the system. At the DCM level, the supervisor can interrupt the TMS320C25 at its highest maskable level to indicate that it has placed new control information

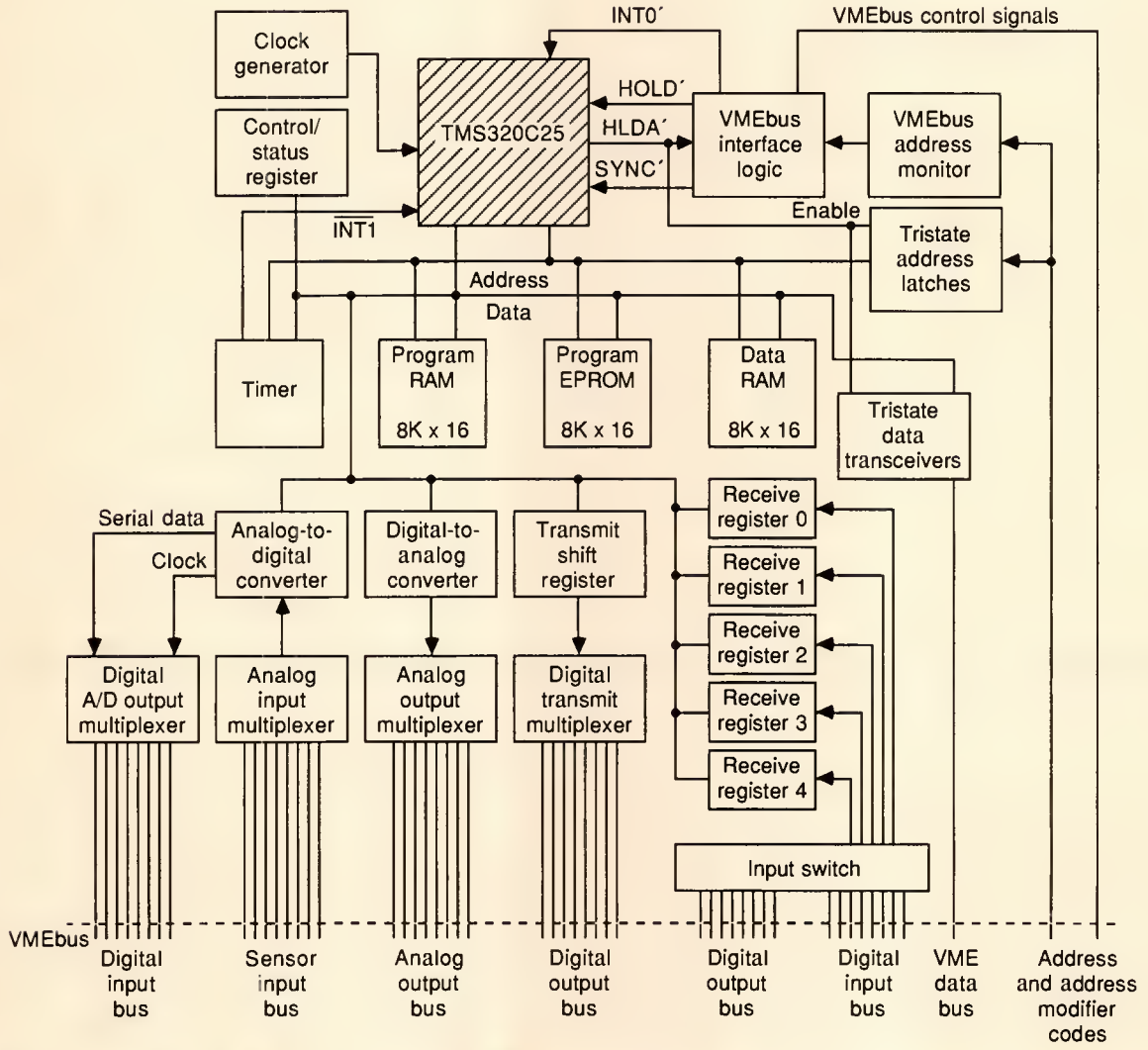


Figure 3. Digital control module architecture.

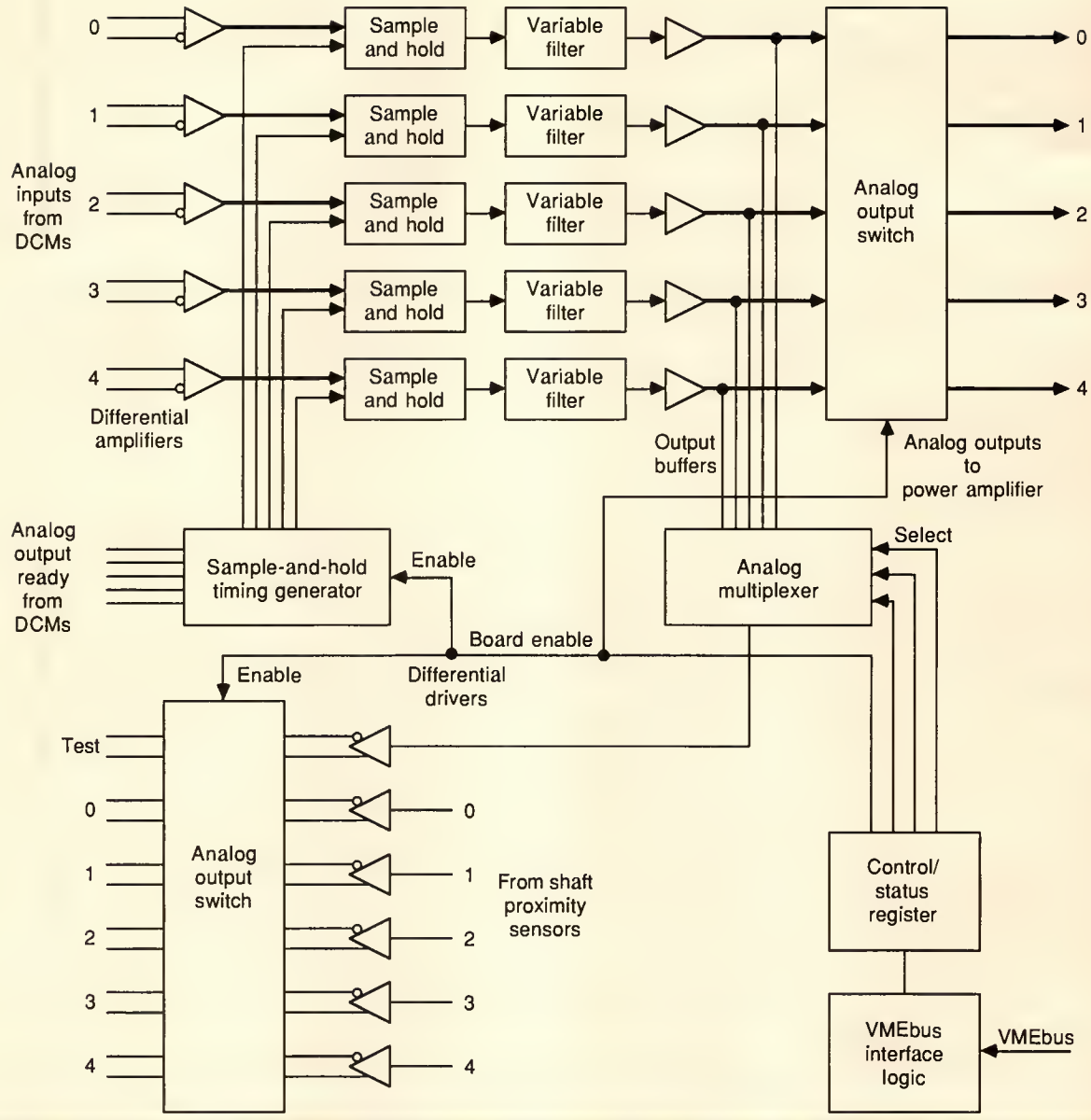


Figure 4. Analog I/O board block diagram.

into the control/status register. The DCMs also interrupt the supervisor if a fault is detected by either a watchdog timer or a self-diagnostic routine. To facilitate testing and demonstration of the fault-tolerant capabilities of the system, each DCM contains a front-panel switch that can be used to simulate a board failure.

Each DCM also has a three-channel, 16-bit programmable timer. One channel supports the watchdog timer function, one interrupts the TMS320C25 at the beginning of each sampling period, and the third is uncommitted. The timer internal to the TMS320C25 is also unused in the current design.

Even with the slower TMS32020, a multiplexed, five-channel, proportional-derivative control algorithm could be implemented at sampling rates of 100 kHz, or 20 kHz per axis. This rate meets the original design requirement. Using the faster chip would at least double the sampling rate to a total of 200 kHz when operating in multiplexed mode. Note that substantially higher sampling rates can be achieved when the DCM is not multiplexed, because no overhead is incurred in context switching. In simplex mode, the DCM can operate currently at a 250-kHz sampling rate. With the use of the faster CPU chip, this rate would increase to the 330-kHz upper limit imposed by the settling times of the analog system components.

Analog I/O board design

Designing a high-performance analog module to be used in an environment as harsh as a VMEbus multiprocessor system is quite difficult. By far, the biggest problems encountered are the noise pickup due to electromagnetic coupling between analog and digital signal lines, noise introduced into the analog circuitry through the power supplies, and noise resulting from the nonzero impedance of ground return paths. We designed the analog switch module with this harsh environment in mind, and we obtained satisfactory results.

The analog I/O module serves several functions in this system. As already stated, it stores the analog outputs on each analog bus as they become available. It further provides filtering of the reconstructed analog control information output. Also, the analog I/O module incorporates differential analog line drivers to drive the analog input buses connected to the DCM inputs. Finally, this module provides the capability to test the analog components in the system effectively. A block diagram of the analog I/O module appears in Figure 4.

The analog I/O module has a simplified VMEbus interface circuit. Each module simply consists of an address monitor, a DTACK´ generator, and a control/status register. When a valid board address is received, the module gives access of its control/status register to the VMEbus master. This register consists of only four active bits: three for address information for the test multiplexer and one to enable or disable the board. When disabled, the board disconnects its output drivers to prevent bus contention. Thus, the system supervisor can enable or disable the analog I/O module and, if active, designate which channel is currently in loopback mode.

A dedicated analog test channel facilitates analog component testing. By setting the address bits in the control/status register appropriately, the supervisor can place any analog channel in loopback mode. Under normal conditions, this channel is assigned to the one DCM that is off line for testing. The DCM under test drives its assigned output bus with a waveform that is sent to the analog I/O board. Here, the waveform moves to the test input bus to be reconverted to digital format by the DCM's A/D converter.

We wanted to reduce the level of noise induced by coupling between the analog signal lines and other signals in the system such as logic signals, 60-Hz power, and interference produced by the switching power supply. So, we designed all of the analog buses to be differential, with maximum signal swings of 20 volts peak to peak. Noise appears the same on both of the differential lines, and this noise can be rejected with high common-mode-rejection-ratio, differential amplifiers at both the DCM and the analog I/O module input stages. Clearly, making the analog signal excursions as large as possible and using differential amplifiers to receive all analog signals greatly reduces coupled noise. Further, we devoted considerable attention to implementation of the analog I/O bus in the actual system. We constructed these lines as twisted pairs, in the shortest length possible. We also placed the analog signal lines as far as possible from digital signal lines, and where crossing was unavoidable, crossed them at right angles.

We also identified and avoided other sources of noise pickup. The power source for the analog circuitry on the DCMs and the analog I/O module is not the VMEbus +12-volt and –12-volt power supplies. Both are switching power supplies and contain an unacceptably high amount of noise that would be difficult to reduce to reasonable levels. Instead of using this power supply, we added a low-noise linear power supply to power the analog system components. Finally, we gave significant attention to proper grounding practices.

We measured performance of the analog system components and found that these efforts are indeed effective in reducing the system noise level. When operating in multiplexed mode and servicing all five channels, the signal-to-noise ratio from the proximity sensor inputs to system output is 68 decibels. With a 12-bit-accuracy A/D converter, systems experience the maximum theoretical signal-to-noise ratio of –72 dB. Thus, our system loses less than one least significant bit of accuracy.

Crosstalk between analog channels when a DCM is operated in multiplexed mode also caused us great concern. Again, with one DCM multiplexed among all five analog channels, we applied a full-scale, 1-kHz sinusoid to the proximity probe input while all other inputs were shorted to ground. In measuring the outputs of the other channels, we found the crosstalk to be less than –72 dB.

Fault-tolerance evaluation

We considered the performance of the system in terms of its fault tolerance, and extensive computer simulations of system reliability models verified that our design satisfies all of the reliability requirements.

We used the US Department of Defense MIL-HDBK-217C formula for estimating individual failure rates of digital integrated circuits for all of the boards in the system.[6] The failure rate of each board is simply the sum of the failure rates for all of its components plus a factor taking into account connector and solder joint failures and the like. The estimated failure rate for each component in the system is

$$\text{DCM failure rate} = \lambda_d = 1.5 \times 10^{-4} \text{ failures per hour}$$

$$\text{Supervisor failure rate} = \lambda_s = 5.0 \times 10^{-4} \text{ failures per hour}$$

$$\text{Analog I/O board failure rate} = \lambda_a = 1.0 \times 10^{-5} \text{ failures per hour}$$

If a system were constructed without incorporating fault tolerance, the system failure rate would simply be

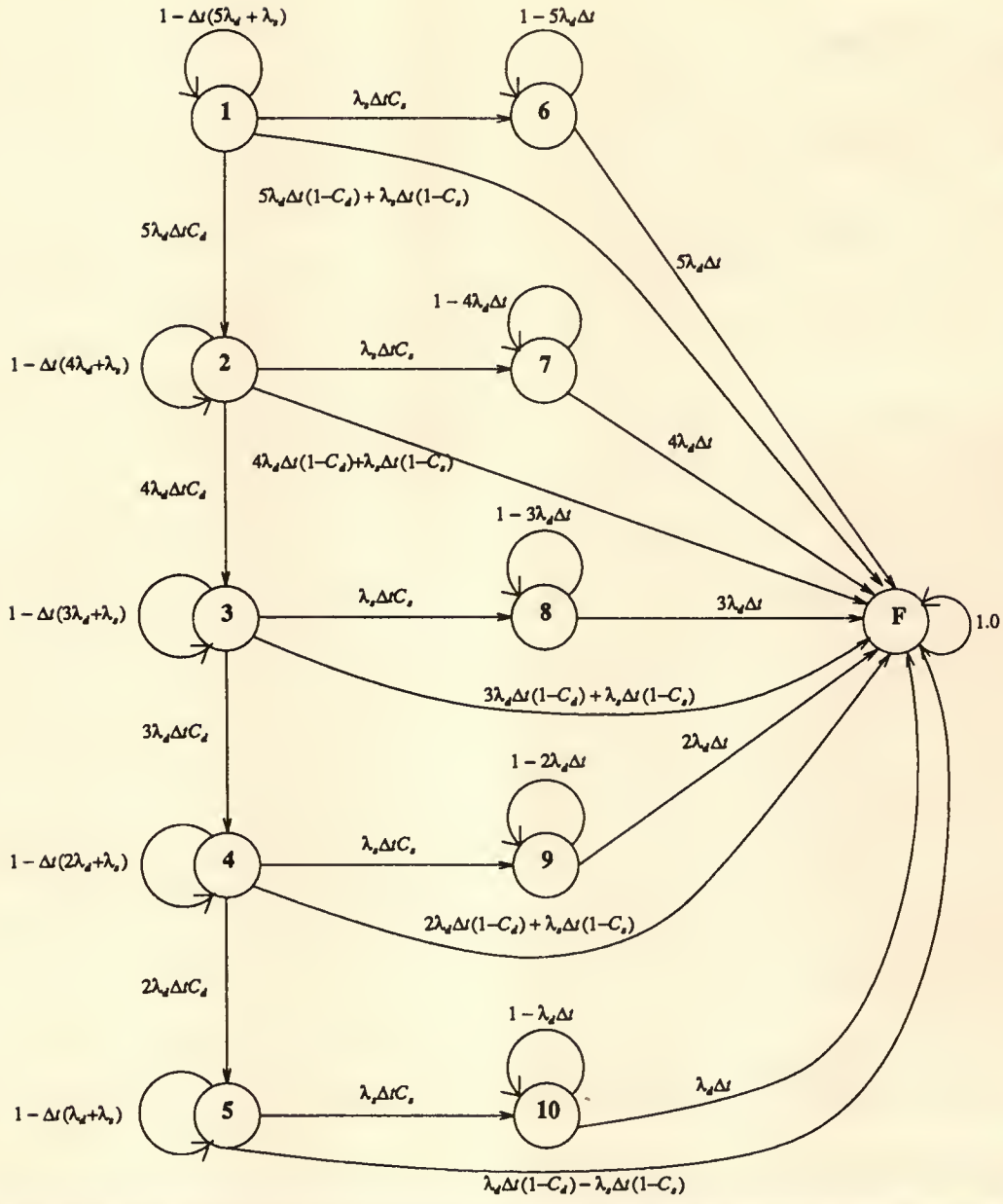


Figure 5. Markov model for system fault tolerance.

the sum of the failure rates of the individual system components. Thus, for a simplex magnetic bearing controller composed of five DCMs, one supervisor, and one analog I/O board:

$$\text{Simplex system failure rate} = \lambda_{\text{simplex}} = 5\lambda_d + \lambda_s + \lambda_a$$

$$\lambda_{\text{simpex}} = 1.225 \times 10^{-3} \text{ failures per hour}$$

Since

$$\text{MTTF}_{\text{simplex}} = \frac{1}{\lambda_{\text{simplex}}}$$

the simplex controller MTTF is

$$\text{MTTF}_{\text{simplex}} = 800 \text{ hours}$$

Thus, the MTTF for a simplex magnetic bearing controller is very poor and does not come close to meeting the original design requirements.

To increase the overall system reliability, we used a software-intensive fault-tolerance algorithm. The responsibility for managing the fault tolerance of the system rests with the supervisor, since each DCM is in a highly time-critical processing loop. The supervisor

assigns processors to individual tasks. Upon request of the system supervisor, each DCM can be switched off line to allow it to enter a comprehensive fault-detection program after the channel it was controlling has been reassigned to another DCM. If there are fewer than five other functional DCMs remaining in the system, one or more must operate in the multiplexed mode, with either a simpler control algorithm or with a significant reduction in sampling rate. A watchdog timer on each DCM detects the failure of the DSP chip. The Intel 8254-2 timer operates at a 5-MHz clock cycle frequency, so watchdog time-out periods can be programmed from 200 nanoseconds to over 13 milliseconds.

The system supervisor also executes fault-detection software. Routines at this level consist of capability checks of the CPU, memory tests, and I/O device tests. The supervisor includes a watchdog timer and a VMEbus timer to detect VMEbus accesses to unassigned addresses and modules that are unable to respond to a bus access due to a module failure. If a redundant supervisor is not included, a supervisor failure leaves the remaining system with no means of tolerating further module failures. Adding redundant supervisors avoids this problem, but cost constraints do not make it a very attractive alternative. Therefore, we analyzed system reliability using only one supervisor module.

Using the estimated failure rates, we constructed a discrete-time Markov model to analyze the effectiveness of the fault tolerance implemented in the system.[7] The Markov model consists of a set of *states* and *state transitions*. Each state represents a certain combination of operational and failed modules, and the state transitions define the state changes that may occur as modules fail. Associated with each state transition is a probability that the transition will occur within a specified time period.

In this application, the states correspond to combinations of failed DCMs, supervisors, and analog I/O modules. However, to simplify the model substantially, we lumped together analog I/O module and DCM failures. This combination results in a reasonable failure approximation for several reasons. For example, the failure rate of the analog I/O module is several orders of magnitude less than that of any of the other modules in the system. Also, the implementation of the DCM fault-diagnostic routines yields DCM analog component failures and analog I/O board failures that are indistinguishable.

Figure 5 shows the Markov model for one example fault-tolerance algorithm. In this algorithm, as soon as a DCM failure is detected, the task assigned to the failed module moves to another DCM. If there were no DCMs in the system that are unassigned to a channel, one of the operational DCMs would have to be multiplexed to accommodate its additional task. States 1 through 5 in the model correspond to 0 to 4 failed DCMs, respectively. When a detected supervisor failure occurs, the system enters states 6 through 10, depending on how many failed DCMs are present. Once the supervisor fails, any additional failure drives the system into state F, which represents a catastrophic system failure. Likewise, any undetected module failure drives the system into state F because of the unpredictable behavior of the failed modules.

To increase the overall system reliability, we used a software-intensive fault-tolerance algorithm.

We simulated the Markov model by computer for varying DCM and supervisor fault-coverage factors. We define fault coverage as the probability that a fault will be detected by some diagnostic technique. The Markov model can be expressed in matrix form as:

$$\mathbf{P}(t + \Delta t) = A\mathbf{P}(t)$$

when $\mathbf{P}$ is a time-dependent vector of probabilities that the system will be in the different states at specific times. The matrix A represents the parameters of the Markov model. This equation tells how the probabilities change as a function of time.

In terms of the Markov model, the reliability of the system at any given instant of time is simply the sum of the probabilities that the system is not in the failed state. More precisely,

$$R(t) = \sum_{i=1}^{10} p_i(t)$$

The Markov model for an N-state system is composed of N first-order difference equations. These equations have been solved through simulations by choosing Δt to be very small and iterating through the equations for the amount of system operating time of interest.

Tables 1 and 2 on the next page list the simulation results for two different cases. In Table 1 we show the supervisor fault-coverage factor C_s fixed at 0.96 while the DCM fault coverage factor C_d varies from 0.95 to 1.00. Similarly, Table 2 lists the results for C_d fixed at 0.97 with C_s varying from 0.95 to 1.00. In both cases, the results covered an operational period of one hour.

It is also possible to determine several other parameters of interest. First, we determine the system MTTF by integrating the reliability function:

$$\text{MTTF} = \int_0^\infty R\,(t)\,dt$$

For the example fault-tolerance algorithm, the overall system MTTF is only 5,100 hours, which is substantially less than the 40,000-hour specification.

To increase both MTTF and short-term reliability, we can compare DCM inputs and outputs in a technique called *pair and spare*. The serial intercommunication network exchanges control information among the

Table 1.
Reliability with $C_s = 0.96$.

Reliability data	
C_d	Reliability
0.95	0.999960
0.96	0.999965
0.97	0.999970
0.98	0.999975
0.99	0.999980
1.00	0.999985

Table 2.
Reliability with $C_d = 0.97$.

Reliability data	
C_s	Reliability
0.95	0.999947
0.96	0.999955
0.97	0.999962
0.98	0.999970
0.99	0.999977
1.00	0.999990

DCMs. Preliminary investigations of the pair-and-spare technique yielded an MTTF estimate of 40,000 hours, which satisfies the original specification.

In addition, it is possible to determine the *steady-state availability* of the system, which we define as the probability that the system will be operational at any given instant of time. Expressed differently,

$$A = \frac{MTTF}{MTTF + MTTR}$$

where MTTR is the mean time to repair. Assuming that most system repairs involve simple module replacements and require 0.2 hours to perform, the steady-state availability of this system is 0.999995.

Clearly, from the Markov model we see that the most significant source of entire system failures is undetected DCM failures. We directed much effort toward increasing the DCM fault-coverage factor. To achieve this increase, we must test each DCM frequently. While it is off line for testing, each DCM also checks its capability for communicating with the other DCMs, and each checks the inputs and outputs from the other DCMs. If these physical variables are outside reasonable limits, we see that the rotor has exceeded its maximum allowed deviation from center, implying that one of the DCMs is malfunctioning. In such a case, the DCM alerts the system supervisor to the presence of a problem so that reconfiguration can commence.

Diagnostic routines executed by the DCM boards include arithmetic unit checks, memory tests, and tests of analog components. The entire analog system can be tested quite effectively. We test the analog system by digitally generating a sinusoidal waveform, feeding the D/A output back to the A/D input, and performing a fast Fourier transform (FFT) on the converted input data.[8] During this test, the analog switch board stores each output of the DCM under test and feeds it back to the DCM over an analog bus devoted to this self-test. In this way, we can test the entire controller, including all analog components.

By examining the spectral content of the input waveform, we easily detect gross faults such as open circuits and failure of the analog power supply. However, the real usefulness in this technique lies in its capability for identifying nonlinearities in analog components. In most cases, analog component failures do not result in stuck-at faults. Most analog failures are partial in nature and stem from causes such as defective resistors and capacitors and partially defective operational amplifiers. If an analog component fails in such a way that it introduces a nonlinearity into the signal path, the failure manifests itself by the presence of higher order harmonics in the FFT. We can also use DC voltages to test analog components using a similar scheme, except that we do not need to perform an FFT. This technique executes much more quickly than a dynamic test, but it is less effective at detecting analog component faults.

One critical factor in determining system reliability is the power supply. A typical switching power supply, such as that used in our VMEbus card cage, has a failure rate of 3.6×10^{-6} failures per hour.[9] Obviously, the system failure rate is, in reality, no less than this figure. However, incorporation of an uninterruptible power supply will ensure an uninterrupted AC power to the system, and there are methods for designing redundant power supplies into the system.

We described a system design that satisfies all of our desired specifications. With a 100-ns multiply-accumulate time, the TMS320C25 DSP chip provides sufficient computational power to achieve a sampling rate of 200 kHz per axis under normal operating conditions. Even when multiplexed among all five axes of control, the system provides sampling rates of 40 kHz per axis when using simple proportional-derivative controller algorithms.

In addition to satisfying the throughput requirements, our system achieves an MTTF rate that is the same as that of a simple analog controller. We obtained these results

without a redundant system coordinator. In many applications, sufficient time may exist to perform voting procedures on the control inputs and outputs to increase the reliability even further. Thus, it should be possible to increase the short-term system reliability and MTTF rate significantly by employing a more extensive software fault-masking and fault-detection algorithm.

We described a unique fault-tolerant multiprocessor architecture intended for real-time, high-speed digital control of a magnetic bearing. The specific application for this controller drove the design to a great extent. But the basic architecture should prove useful for many real-time distributed control applications.

Acknowledgment

The United States Army Research Office and the Exxon Education Foundation supported this work in part.

References

1. R. Filman and D. Friedman, *Coordinated Computing*, McGraw Hill Book Co., New York, 1984, 370 pp.
2. A. Carmichael, S. Hinchliffe, P. Murgatroyd, and I. Williams, "Magnetic Suspension Systems with Digital Controllers," *Review of Scientific Instruments*, Vol. 58, No. 8, Aug. 1986, pp. 1611-1615.
3. M. Scudiere, R. Willems, and G. Gilles, "Digital Control for a Magnetic Suspension System," *Review of Scientific Instruments*, Vol. 58, No. 8, Aug. 1986, pp. 1616-1626.
4. J.H. Wensley et al., "SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control," *Proc. IEEE*, Vol. 66, No. 10, Oct. 1978, pp. 1240-1255.
5. J. Wensley and C. Harclerode, "Programmable Control of a Chemical Reactor Using a Fault-Tolerant Computer," *IEEE Trans. Industrial Electronics*, Vol. IE-29, No. 4, Nov. 1982, pp. 258-264.
6. *MIL-HDBK-217C Notice 1, Military Standardization Handbook: Reliability Prediction of Electronic Equipment*, US Department of Defense, May 1980.
7. M. Shooman, *Probabilistic Reliability, An Engineering Approach*, McGraw Hill, 1968.
8. M.V. Bossche, J. Schoukens, and J. Renneboog, "Dynamic Testing and Diagnostics of A/D Converters," *IEEE Trans. Circuits and Systems*, Vol. CAS-33, No. 8, Aug. 1986, pp. 775-785.
9. "*MTBF* and Failure Rate Calculations," *Technical Manual and Data Book: STD Bus, Systems and PROM Programmers*, PRO-LOG Corp., New York, Nov. 1985, pp. 4.1-4.2i.

Steven W. Yates works with the OEM platform operation at Intel Corp. in Hillsboro, Oregon. From 1987 to 1988 he was a research assistant with the Rotating Machinery and Controls (ROMAC) Laboratory at the University of Virginia. His current interests include advanced microprocessor systems, fault-tolerant computing, and parallel processing.

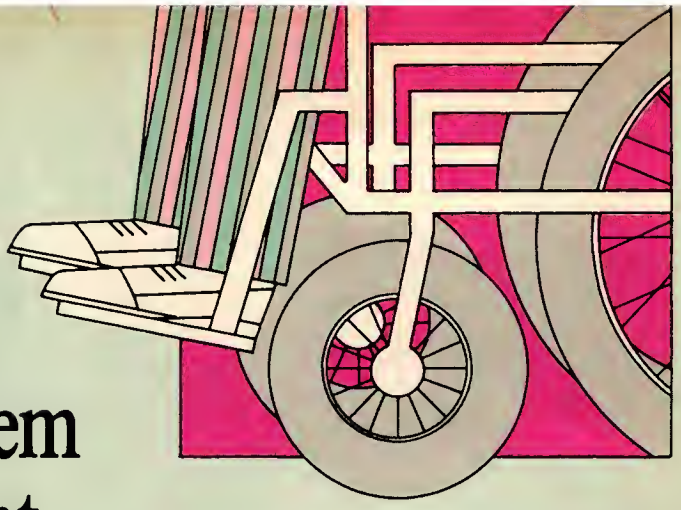
Yates received BS and MS degrees in electrical engineering from the University of Virginia. He is a member of the Computer Society.

Ronald D. Williams is an assistant professor in the department of electrical engineering at the University of Virginia. He is also a principal faculty member in the university's Center for Semicustom Integrated Systems. His research interests include design for testability and reliability and digital system design for signal processing and control applications.

Williams received the BS and MS degrees from the University of Virginia and the PhD degree from the Massachusetts Institute of Technology, all in electrical engineering. He is a member of the IEEE, the Computer Society, Tau Beta Pi, Eta Kappa Nu, Omicron Delta Kappa, and Sigma Xi. He currently serves as chairman of the Computer Society's membership development committee and has served as finance chairman of the society's Conferences and Tutorials Board.

Questions concerning this article can be addressed to Ronald D. Williams, Center for Semicustom Integrated Systems, University of Virginia, Charlottesville, VA 22901.

FEATURE



An Operating System for a Fault-Tolerant Multiprocessor Controller

This approach meets the needs of many real-time control applications. Self-testing and automatic reconfiguration permit graceful degradation in response to faults.

Ronald D. Williams
Barry W. Johnson
Thomas E. Roberts

University of Virginia

Users in an increasing number of fields apply microprocessor-based controllers to tasks because of their numerous advantageous features. These features include flexibility, computational capability, low power consumption, and small physical size.[1-4] Users employ microprocessor-based systems to control robots, aircraft flight control systems, process controllers, and prosthetic devices—to give just a few examples.

In many cases users depend on microprocessors whenever they need system integrity and correct performance in a controller. As a result, they require attributes such as fault tolerance, high reliability, high availability, and safety.

In general, we define fault tolerance as the capability of a system to perform its desired functions correctly in the presence of either hardware failures or software errors.[5-7] To evaluate the capability of a given design to achieve correct operation, we use metrics of reliability, availability, and safety. Reliability, for example, is the probability that a system will operate correctly throughout an interval of time, given that it was operating correctly at the beginning of the time interval. Availability, on the other hand, is the probability that a system is operating correctly at an instant of time. Finally, safety is the probability that a system will either perform correctly or will fail safely.[6]

The reliability, availability, and safety requirements placed on a system depend on whether the functions performed are critical to life, critical to a mission, or for convenience.[8] Life-critical functions, if performed incorrectly, can result in the loss of life, equipment, or both. Mission-critical functions are those necessary to perform certain tasks mandatory for the completion of a given job, while convenience functions enhance a system but are expendable. Fly-by-wire flight control exemplifies a life-critical function, while certain communications operations and electronic diagnostic capabilities might be considered mission-critical and convenience functions in certain applications.

Numerous applications require fault tolerance, reliability, safety, and availability. One specific, real-time control application in which the use of a microprocessor-based controller is beneficial, and processing integrity in the form of fault tolerance is a necessity, is the electric wheelchair. The control problem associated with an electric wheelchair possesses a number of attributes characteristic of many real-time control applications. In the electric wheelchair, microprocessor-based control can provide several desirable

0272-1732/88/0800-0018$01.00 © 1988 IEEE

features,[9] which are either difficult or nearly impossible to provide with a conventional analog controller. Such features include adaptive closed-loop control,[10] digital filtering of the command inputs, and an on-line battery-monitoring system.[11] In addition, the use of the microprocessor increases the flexibility of the system such that modifications necessary to tailor the wheelchair to a particular user are easily completed.

The design of a wheelchair controller also must address major concerns for the reliability, availability, and safety of the resultant system. For example, a surge in the power supply of an electric wheelchair may cause the processor to lock up, and, as a result, the wheelchair may run uncontrollably. In addition, permanent hardware faults within the processor can lead to undesirable responses. Finally, users of wheelchairs depend on the system being operational a large percentage of the time to meet their transportation needs.

To achieve the desired reliability, availability, and safety, designers must take special care in designing the controller. The system design may include the addition of hardware, and the careful preparation of software, to improve the capability of the system to keep running, safely shut down after a fault occurs in the microprocessor equipment, and/or locate a fault for diagnostic purposes.

We focus on the development of an operating system that is a central component of a fault-tolerant multiprocessor. While we designed the operating system to meet the needs of the electric wheelchair application, we feel the resulting system and concepts can apply to a variety of real-time control systems. The operating system, while relatively simple and small, supports multitasking and multiprocessing. The system supports both self-diagnostics and cross-diagnostics for fault detection. In the event of a fault, the system permits rapid reconfiguration in a manner that retains processing for the highest priority tasks. Since the hardware needed to provide fault tolerance is available when there are no faults, the operating system can exploit this excess capacity to accomplish lower priority tasks during normal operation. This approach yields graceful degradation in response to faults in the system components.

Background

Figure 1 displays the structure of the electric wheelchair system, which consists of seven essential components. These are:

- user command sensors,
- controller electronics,
- motor drive electronics,
- motor feedback sensors,
- electric motors,
- transmission, and
- wheelchair chassis.

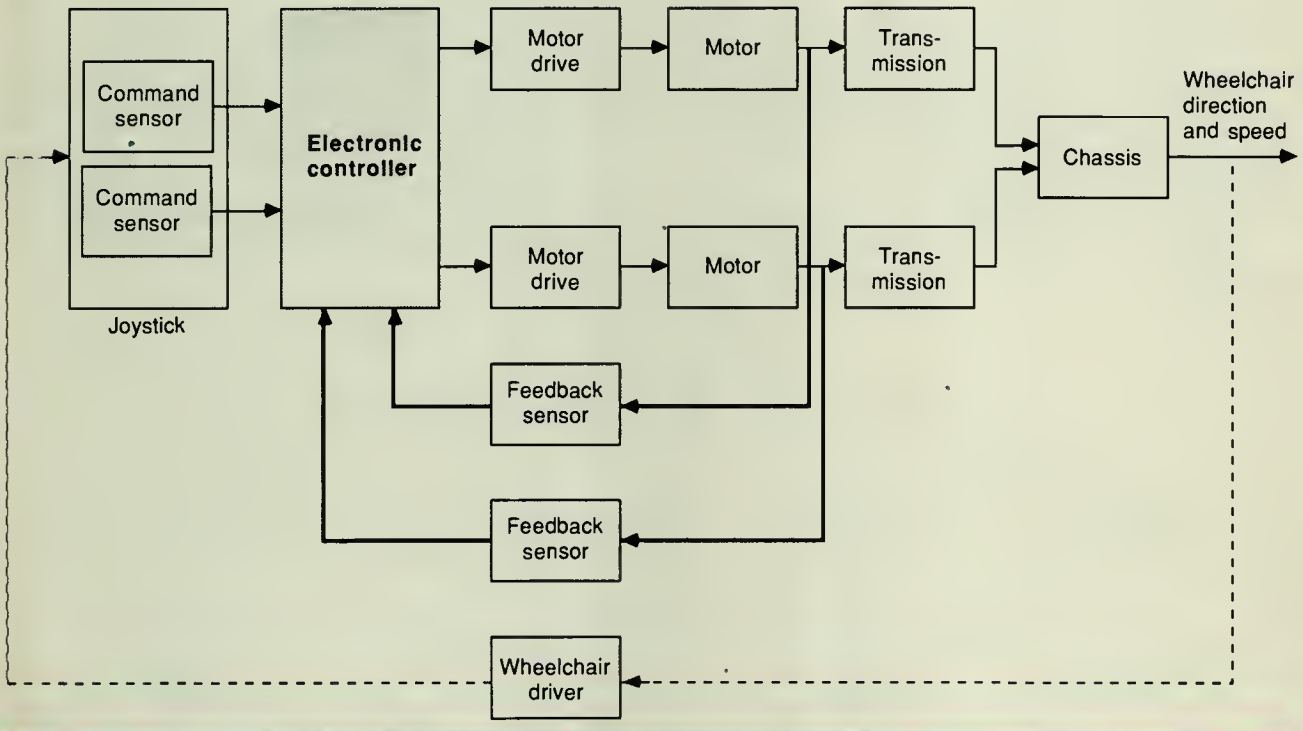


Figure 1. Block diagram of the electric wheelchair system.

The command sensors vary from wheelchair to wheelchair but may include, for example, traditional joysticks, tone detectors, voice recognition units, and sip-and-puff sensors. (These last sensors allow the user to direct wheelchair movement and velocity by blowing into a tube.)

The controller electronics use the driver commands and the feedback signals to develop control signals for the motor drive electronics. The motor drives typically are transistor-bridge circuits[2] that control the magnitude and direction of current flow to the wheelchair's direct-current (DC) motors. The motor feedback sensors may be tachometers, back electromagnetomotive-force (EMF) sensors,[2] or other motor speed detection devices. The DC motors are usually small, permanent-magnet ones, and the power source for the wheelchair typically consists of two batteries similar to those found in automobiles.

The structure of the electric wheelchair system is common to many real-time control applications. An electronic controller processes user input commands and system feedback signals. The controller then produces commands to some type of motor or actuator that drives the system in the correct direction and at the correct speed. The core of this type of system, the electronic controller, performs any processing that is required. For example, the feedback or command signals may require filtering to remove unwanted, and possibly detrimental, noise. Also, the controller electronics perform any velocity or position feedback to be implemented.

The goal of our work was to develop a fault-tolerant multiprocessor to perform the functions of the electronic controller. We required fault tolerance to maintain safe operation in the presence of faults. We desired multiprocessing to provide increased computational capability for the performance of more advanced and flexible functions in the controller.

Figure 2 shows the architecture of the fault-tolerant multiprocessor controller. The architecture is similar to that found in the Tandem NonStop[12] and the Synapse N+1 systems.[13] The system uses a shared-memory, multiprocessor architecture, which can support the parallel execution of tasks and the redistribution of the tasks in the event of a failure of one or more processors. Each independent processor has its own program and data memory (called the *local memory*). The processors communicate via the shared memory. Each processor can perform any of the tasks required of the system; therefore, a single, spare processor can substitute for any processor that fails. Each processor stores all of the programs in local memory such that the redistribution of processing tasks can be performed by simply specifying the appropriate tasks for each processor to execute.

We can produce a reliable and safe fault-tolerant controller by including software and hardware to detect faults and reconfigure the system.[14] We basically accomplish this using redundancy and self-diagnostics. Each microprocessor includes self-diagnostic software that runs concurrently with the normal control operation. We can design self-diagnostics to detect a large percentage of faults in the control circuitry that can affect the control process. The multitasking capabilities of the operating system simplify the inclusion of diagnostic software for all control system components, both inside and outside of the computer system.

The general architecture described in Figure 2 supports several possible configurations, which can provide, at a minimum, fault detection for the digital controller. One possible configuration, known as the *simplex* architecture, simply consists of a single microprocessor system that includes self-diagnostic software to test itself concurrently with normal operation. The simplex configuration is the simplest possible microprocessor-based configuration, but it has limited fault-detection capability and no fault tolerance. The simplex configuration can be used, however, as a baseline for measuring the improvement attained with the inclusion of additional hardware and software for redundancy.

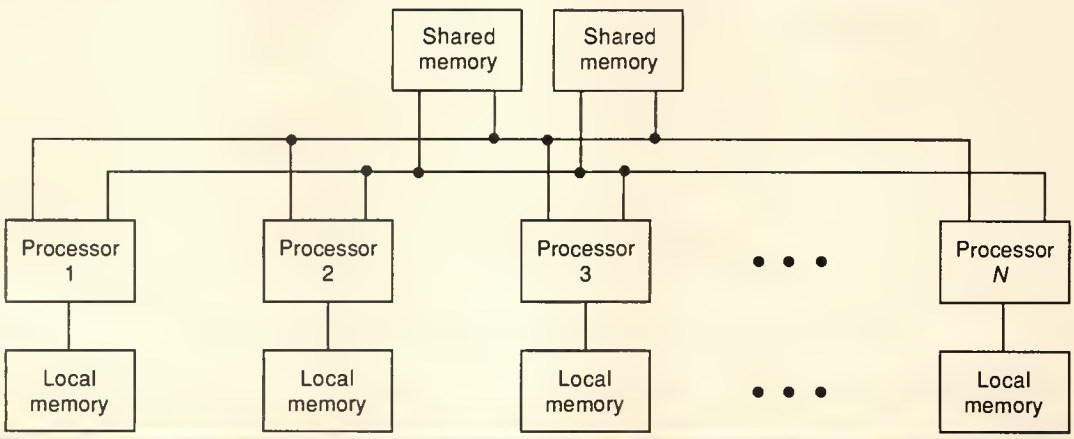


Figure 2. Architecture of the fault-tolerant multiprocessor.

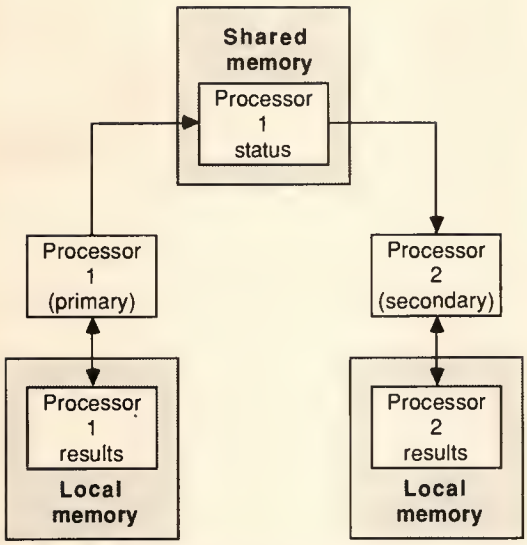


Figure 3. Conceptual implementation of standby sparing in the multiprocessor.

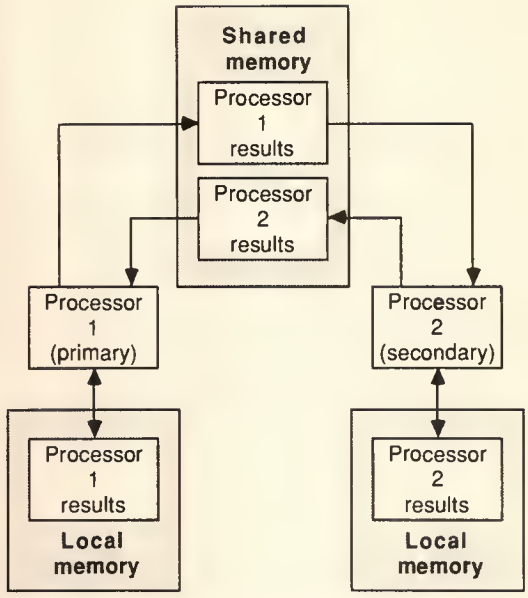


Figure 4. Conceptual implementation of the full-duplex approach.

A second possible configuration, known as *standby sparing,* consists of two microprocessors with each performing self-diagnostics. One microprocessor (the primary processor) controls action, while the other (the secondary processor) is a standby spare. If the primary processor fails and detects its own failure, it signals the secondary processor to assume control. The system then runs as a simplex system. Simplex operation also results if the secondary processor fails and detects its failure. As mentioned earlier, communication between the two processors in the standby-sparing system occurs via the shared memory. The secondary processor monitors a specific location in the shared memory and assumes control of the system in the event that the primary processor places a specific value in the location. The key feature of this form of standby sparing is that the secondary processor assumes control only when told to do so.

We illustrate the concept of the shared-memory implementation of standby sparing in Figure 3. Both the primary and secondary processors have access to their own local memory for program execution, self-diagnostics, and temporary data storage. The secondary processor continually monitors the shared memory for a command to assume control of the system. If the primary processor provides such a command, the secondary processor assumes control.

A third configuration, known as the *full-duplex* approach, is a variation of the standby-sparing concept and includes two microprocessors, with one (primary) controlling the process and the other (secondary) acting as a spare. However, in the full-duplex approach, the processors perform both self-diagnostics *and* a cross-diagnostic. The cross-diagnostic allows the processors to check each other's operation. The full-duplex method requires each of the two microprocessors to provide data regarding its status to the other microprocessor. Failure to provide the necessary data indicates a fault. The processors communicate via the shared memory, and each processor expects the other to set repetitively a value in shared memory. The failure to set the appropriate value indicates a fault and allows the secondary processor to assume control of the system or the primary processor to recognize the loss of the secondary processor.

The full-duplex method offers two significant improvements over standard standby sparing. First, we can detect a runaway failure of the primary processor in the full-duplex configuration because of the cross-diagnostics. Traditional implementations of standby sparing are susceptible to a runaway processor failure because the secondary processor assumes control only when commanded by the primary processor. The cross-diagnostics in the full-duplex system detect the problem, and the fault-free microprocessor transfers control.

The second advantage of the full-duplex approach is that the primary processor will not attempt to transfer control to a failed secondary processor. The possibility exists in the standby-sparing scheme for the secondary processor to have a fault undetected by its self-diagnostics. If the primary processor were to have a subsequent fault, control would be transferred to the faulty secondary processor, causing unpredictable control action. In the full-duplex option this is less likely since the primary processor verifies that the secondary is successfully performing its self-diagnostics.

Figure 4 shows the conceptual implementation of the full-duplex approach in the shared-memory system. Both processors write results concerning their statuses

Standby sparing consists of two microprocessors, each with self-diagnostics. One controls the action; the other is a standby spare.

to the shared memory where both processors can read the results. Consequently, the processors periodically check on each other.

A fourth configuration, known as *reconfigurable duplication,* allows both the primary processor and the secondary to perform all of the control system calculations, as well as self-diagnostics. The two processors compare all control calculations performed by each processor. Once again, Figure 4 can be used to illustrate the concept. Both processors place the results of their calculations in both shared memory and local memory. Each processor then reads the other's results from shared memory and compares them to its own. Differences between the two results indicate a fault. If one processor fails and detects that failure via its own self-diagnostics, the system reverts to simplex operation. If one processor fails but does not detect the failure, a comparison mismatch usually exists between the secondary and the primary processor. Since both processors consider themselves fully operational, the system shuts itself down. If one processor fails without detecting the failure and no comparison mismatch exists, the system would fail in an unpredictable manner.

Each of the alternative configurations can be used to provide some level of fault-tolerant control and can include operability checks of the ancillary components, as well as the microprocessor itself. We compared the options extensively using modeling techniques of reliability, safety, and availability.[14] The multiprocessor architecture in Figure 2 supports any of the four alternatives just described. Our objective was to select an approach and then develop a software system that could implement the technique using the basic hardware configuration of Figure 2.

Prototype hardware structure

We considered each of the alternative configurations and found the full-duplex option to be most reliable without compromising safety and availability.[15] Consequently, we selected this option for further study and prototype implementation. We also selected the full-duplex architecture because it can easily be modified into a reconfigurable-duplication system. Therefore, the laboratory system can easily implement two approaches to fault-tolerant control. Once selection was final, we found it necessary to develop an operating system that could provide the necessary operation. Figure 5 shows the prototype hardware we implemented in the laboratory. It includes two processors (each with its own local memory and input/output capability), a shared (global) memory, a shared (global) bus, and an IBM personal computer for program development and experimental evaluation. Figure 5 also includes two universal asynchronous receiver transmitters for communicating between each of the two processors and the IBM computer. Figure 6 is a photograph of the laboratory prototype.

The processor boards in the prototype system use the 8088 microprocessor and can coordinate access with other processors to a common bus. The processor boards also include several additional components such as an 8253 counter/timer, 8259 interrupt controller, and memory. The 8253 three-channel counter/timer produces regular interrupts at the required sampling intervals of the control system and measures the time required to process a given function. The 8259 programmable interrupt controller manages the different sources of interrupts and provides the necessary interrupt vectors to the 8088 microprocessor. Some read-only memory (ROM) normally contains the program code; however, in the prototype it contains a monitor that was used for system development. Finally, read-write memory (RWM) stores both programs and data. The ROM and RWM on the processor boards constitute the local memory (as distinguished from the shared memory), which is only accessible by the associated processor.

The shared memory resides on a separate board from the processors and connects to a shared bus so that it can be reached by all processors. The prototype system includes only RWM in the shared memory. We configured the address space of the shared memory so that the memory addresses used for shared memory do not overlap with those used by the processors' local memory.

The input/output devices used in the system include analog-to-digital and digital-to-analog converters as well as parallel and serial input/output ports. We designed and added a set of I/O devices to each processor; either processor can completely control the system in the event of an I/O device failure.

The prototype hardware supports the desired fault-tolerant architectures provided that a software system is developed to perform the functions of fault detection and redundancy management. The operating system described in the subsequent sections utilizes the prototype hardware and provides the necessary features to achieve fault-tolerant operation.

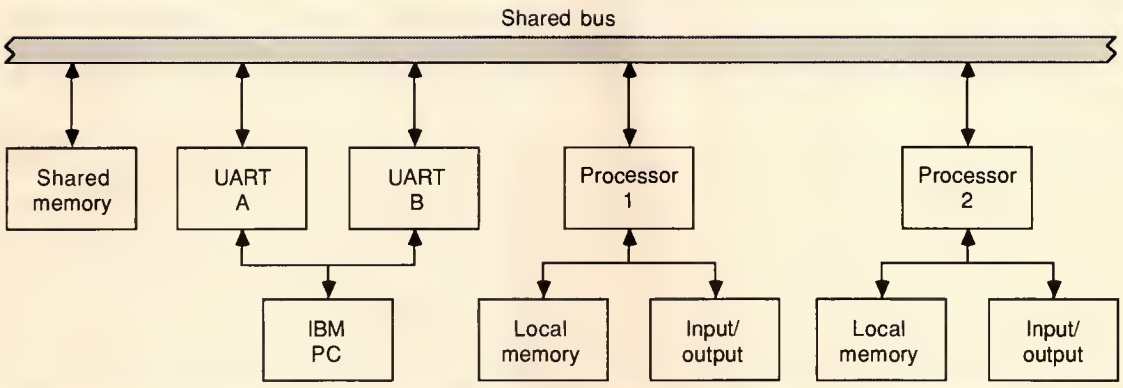


Figure 5. Hardware structure for the prototype system.

The operating system

Operating systems for real-time control must satisfy constraints that can be significantly different from those imposed upon the more-familiar general-purpose operating systems.[16] The sensors and actuators of the controlled process must be serviced at regular intervals. This need imposes time constraints upon the behavior of the real-time operating system that are not necessary in the general-purpose data processing situation. The typically rapid processing demanded for control coupled with the limitations of current microprocessors requires that the controlling software remain resident in the system memory. Scheduling of individual process execution must be accomplished by the operating system with careful consideration of the time constraints

Figure 6. Prototype implementation of the fault-tolerant multiprocessor.

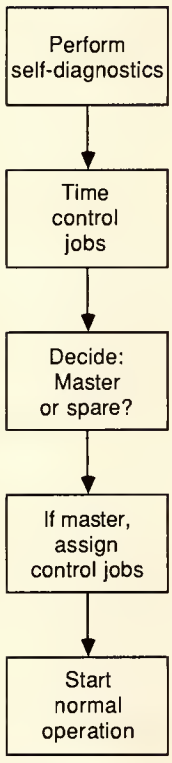


Figure 7. Sequence of operations performed during initialization.

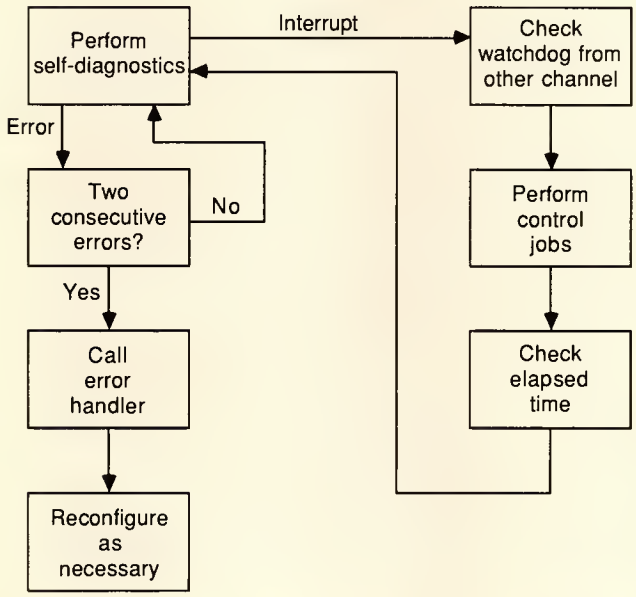


Figure 8. Sequence of operations performed during normal operation.

associated with each process. Coupled control programs necessitate efficient interprocess communication.

Operating systems for real-time control can be simpler, in some respects, than the typical general-purpose system because of its specialization. A significant part of a general-purpose operating system involves the management of a file system, and the process controller we describe needs no files. Static memory assignments simplify the often complex task of memory management for this particular application. Fixed assignments similarly simplify the management of input and output devices. The need to reassign these resources to different processors in the event of a detected fault, however, complicates the otherwise simple approach to memory and I/O device management.

We designed the operating system to be general enough to take care of the fault-tolerance and program sequencing aspects for any sampled-data control application. In addition, we had two fundamental goals beyond the fault-tolerance and performance aspects. Specifically, we desired that the system be capable of using the computing resources of the spare processor while it is in its normal standby status. Also, we wanted to include the capability of expanding the system beyond two processors.

If the computing resources of the spare processor are available, the spare can perform some of the tasks of the control system that are desirable but not essential. For example, the automatic battery-monitoring capability for a wheelchair is nice to have, but it can be done without. Such tasks can be assigned to the spare processor, allowing the controlling processor time to perform more complex control computations. Such lower priority jobs will be lost if the spare needs to take over control of the system, but, since they are low priority, system operation remains satisfactory.

Similarly, a control system may include more essential control calculations than can be performed by one processor within the allowed time. In this case, the jobs cannot be lost as the second processor takes over system control without losing vital control functions. Allowing system expansion by increasing the number of processors without losing the fault tolerance is necessary in this case.

We organized the software to handle three primary operations: initialization, normal operation, and error condition handling. The initialization process consists of performing self-diagnostics on each processor, determining which of the processors is the primary one and which is the spare, and assigning the processing of jobs associated with the wheelchair control algorithms (control jobs) to the primary and/or the spare. During normal operation, the operating system controls the sequencing of both types of jobs (control processing and self-diagnostics). An interrupt service routine controls when the jobs are sequenced, and a dispatcher routine actually performs the sequencing.

Error condition handling occurs following the detection of an error by the self-diagnostics software. Based on the error detected, the action taken can include anything from no action to deliberately shutting down a processor. Figures 7 and 8 contain block diagrams of the software operations during initialization and normal operation.

The primary routines necessary to perform initialization, normal system operation, and error handling are the job scheduler, process scheduler, diagnostic routines, and error handler.[17] The job scheduler executes on the primary processor and determines which jobs will be assigned to execute on each processor. The job scheduler uses the execution times and priorities of the jobs when establishing the assignments. The process scheduler executes on each processor and allocates execution time to each assigned process in a manner that ensures satisfaction of process real-time constraints. In addition to the control jobs, the processors must execute diagnostic routines that test and verify the correct functionality of the various system components. Finally, the error handler supervises the actions taken as a result of faults detected by the diagnostic routines.

The job scheduler. The standard tasks for a job scheduler are to examine all jobs submitted to the system and to select which jobs will be executed. In the case of this multiprocessor system, the job scheduler further selects which processor will execute the job. The job scheduler for this system also includes functions for measuring timing parameters of the available jobs.

After initial hardware synchronization, the job scheduler sets up a table describing all jobs included in the system software. Since either processor can execute any job, the local memory of each processor contains the code for all jobs included in the system. The job table includes the starting address for each job and a descriptor that provides information such as its priority, its autonomy, and its type, as illustrated in Figure 9. The priority information permits the assignment of limited processor resources to handle the most important jobs. Lower priority jobs are assigned to processors only if sufficient processing resources are available. The autonomy information tells the scheduler whether or not the job is dependent upon other jobs. The type may be either control or diagnostic. Duplicate copies of the job table reside in all processors, but we included an additional information item for each job descriptor to identify which processor actually executes each job.

The job scheduler initially submits all diagnostic jobs for execution to determine how much of the system is functioning properly. As this check progresses, the shared and local memories store the operability status of each processor and its ancillary components. After completion of this section of initialization, a word in shared memory indicates the operability status of each processor.

The job scheduler next submits each control job for individual execution. A hardware timer measures the time required to execute each of these jobs, and the measured time is added to the information recorded in the job table. Measurement of the execution time permits intelligent assignment of the control jobs to the available processors. A degree of uncertainty exists with this timing process because different times may be required by different paths through the program. We address this uncertainty by allowing a margin for error in the maximum total time that can be assigned for control processes on any processor. Unlike general-purpose computing, we can estimate this uncertainty for typical control loop programs.

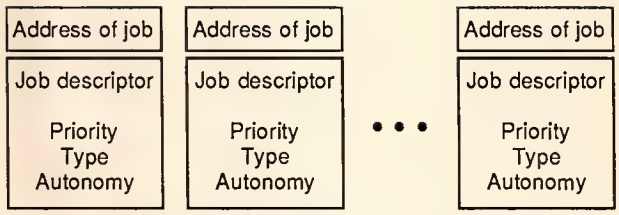


Figure 9. The job table.

Upon completion of the initialization, the job scheduler contains sufficient information to establish job assignments. It designates one of the available processors to serve as the primary processor. At least one other processor should be designated as the secondary processor, and additional processors may be assigned lower designations. The job scheduler combines the priority and timing information recorded in the job table with the various processor designations to establish job assignments.

The job scheduler running on the primary processor decides which of the control jobs the primary processor performs and which the secondary or other processors perform. The scheduler adds up the total time required by all control jobs. If this time is less than a fixed maximum, the scheduler assigns all jobs to the primary processor and the configuration is complete. If the total time is greater than the fixed maximum, the configuration process continues by selectively moving job assignments from the primary to the secondary processor until the total time assigned to the primary processor is less than the fixed maximum. The scheduler then evaluates the total time assigned to the secondary processor to determine whether it exceeds a similar fixed maximum.

Job transfers may occur from the secondary processor if other processors exist, or some jobs may not be assigned to any processor if insufficient processing resources exist. The priorities and independence of the jobs as marked within the job table determine the selection of jobs to be transferred to lower designated processors. Highest priority jobs are scheduled first to ensure their assignment to processors with the highest

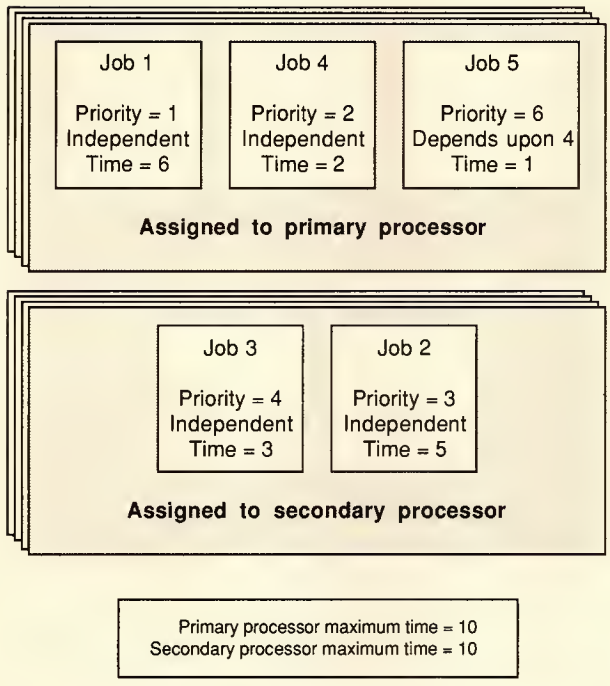


Figure 10. Example job schedule for two processors.

possible designation. The processor autonomy information identifies sets of processes that must be assigned to a single processor, and these sets are not broken across processor boundaries. The job scheduler assigns the highest priority jobs to the primary processor, and some low priority jobs may be assigned to the primary processor because of their dependence upon higher priority jobs. Figure 10 illustrates an example job schedule.

The diagnostic jobs need not be assigned by the job scheduler because they are statically assigned to each of the processors. Each processor must execute certain self-diagnostic and cross-diagnostic processes. The execution times of these diagnostics are fixed and known and contribute to the fixed maximum time available to each processor for the execution of control processes.

The task of the job scheduler is to assign the various offered processes to the available processors. The job scheduler for this real-time operating system differs from job schedulers for a general-purpose operating system because no new jobs are expected to be offered after the initial system configuration. The job scheduler in a general-purpose operating system continually receives new jobs from the environment and schedules these jobs for access to the processor resource. When the job scheduler for this real-time system has completed its initial assignment, it will not be called again unless system problems develop. For example, the job scheduler could be called upon again to create a new job assignment after the loss of a faulty processor.

The process scheduler. After the job tables have been established with the jobs assigned to the appropriate processors, the normal operation of the processors can start. The standard responsibility for a process scheduler is to select which of the assigned processes is to be executed at a given time. In this particular system, processes execute in a sequential fashion with a regular interrupt used to ensure real-time sampling of the controlled environment. The interrupt-driven process scheduler permits concurrent system diagnostics to execute in the *background* while various control programs execute in the *foreground*.

Upon receipt of a timing interrupt, the process scheduler saves the state of the machine, samples all input values, and presents all previously calculated output values to their associated ports. Then the process scheduler selects a control process, restores the state of the machine to that required by the selected control process, and transfers processor control to the process. When the process completes a pass through its algorithm, control of the processor returns to the process scheduler. The process scheduler then selects another control process to run on the processor and continues until all of the control processes have finished. The process scheduler next restores the processor to the state that was present at the time of the last timing interrupt. Processor control then transfers to the background diagnostic tasks. All of the control processes initially assigned by the job scheduler are calculated to require less time than the interval between sampling interrupts. Additional time is allowed to permit execution of the background diagnostic tasks.

A process scheduler operates independently on each processor. The process scheduler is the only resource manager needed in this system because all resources other than the processors are assigned statically. Local- and shared-memory assignments for data are established either statically to support operating system or diagnostic functions or by the job scheduler at the time of job assignment to support particular processes. Particular input and output resources are assigned to specific processes. The only resources subject to dynamic dispatch are the processors, and each of these has its own process scheduler.[18]

Each process uses local memory to contain relevant data, but the process scheduler ensures that certain critical process data is also copied into designated shared-memory regions periodically. The process scheduler also assigns regions of shared memory for interprocessor communication. It places summary results from diagnostic self-tests by each processor in assigned shared-memory locations. All processors can view these locations, therefore providing a mechanism for cross-diagnostic checking of one processor by another. Each processor contains hardware watchdog timers to permit local detection of runaway conditions, and the regular reporting of individual processor status in shared memory provides a second external operational check.

Diagnostic routines. The software of the prototype system contains seven self-diagnostic routines. The routines partially provide the fault-detection capability necessary to support real-time reconfiguration. A side benefit of the routines is the support of fault location to assist in the maintenance process. Self-diagnostics in the prototype system include checks of the local ROM, the local and shared RWM, the processor chip itself, the timer chip, the input sensor-processing circuits, and the output actuator drive circuits. Additional self-diagnostic routines could be included easily to test additional sensors and actuators.

The local RWM test involves writing and reading two separate bit patterns, which are both alternating 1s and 0s and complements of each other. Because the bit patterns are complements, each memory data line must assume both logic values during the test. The test begins by writing one of the bit patterns to the highest address in the memory. The routine then reads the contents of the highest address to ensure that the proper datum was written and can be read properly. Then, in sequence, the test brings each address line low and reads the content of that address to ensure that it does not equal the datum written to the highest address. The procedure then repeats using the second bit pattern. This check detects whether the address lines are stuck either high or low and checks for data lines stuck at a particular value because of the two patterns of alternating 1s and 0s.

Note that this test does not check every location in the memory but, instead, attempts to verify the correctness of the address and data lines. We plan to include traditional parity in a later version of the hardware to provide additional memory test capability. The shared RWM test is identical to that for the local RWM, except that extra precautions must be taken to ensure that two processors do not check the shared memory at the same time.

The local ROM test involves checking the contents of a specific memory location to ensure that it matches what was previously stored there. Then, in sequence, the routine changes each address line's value (high or low) and reads the contents of the corresponding address. It compares this value with the original value brought in and detects an error if the new value is equal to that original value. A starting address, chosen by trial and error in the experimental system, ensures that the values at each of the checked ROM addresses are different. In a production system, the ROM could be "seeded" with data at specific locations to ensure that this test would work.

The processor chip test checks three sections: the flags, the arithmetic logic unit, and the registers. The test completes flag checks by verifying that the flags can be set and reset. It performs the arithmetic logic test by comparing the results of calculations to prestored values. By writing a bit pattern to each register, performing some register shift operations, and comparing the results to prestored values, the routine checks the registers.

The secondary processor correctly assumed control of the system.

Measuring the time required to complete a routine tests the timer. We determine the expected result in advance and compare it with the measured time. A comparison error outside of a specified tolerance flags a timer error.

Loopback tests check the sensor input-processing circuitry. Specifically, the test provides the output of a digital-to-analog converter, or DAC, as the input to one of the analog-to-digital converter, or ADC, circuits. This check permits commands sent out via the DAC to be verified by routing them back through the ADC. A similar approach verifies correct operation of the actuator output circuitry.

None of the seven tests discussed trigger responses on the first detected error. The tests assume each error detected to be transient, and they perform an immediate retest. The test initiates further action only if either two consecutive errors occur or the number of detected transient errors exceeds a specified value. If two consecutive errors occur, the routines assume a fault exists in the component being tested and take remedial action. If the total number of detected transient errors exceeds a specified value, the tests consider the component being tested to be unreliable for use in the system and take action to remove the component from service.

Error handler. When the diagnostic routines determine that a processor has failed or is unreliable, the shared memory is marked with an indication that the particular processor is not available. This shared-memory marking may be accomplished by the self-diagnostics in the failing processor, or the mark may be made by another processor through its cross-diagnostics. In either case, the faulty processor suspends service.

If the faulty processor was designated as the primary processor, the secondary processor will notice on its next pass through the interrupt service routine that the primary processor is not available. The secondary processor will then assume the duties of the primary processor and shed its secondary processor responsibilities. If more than two processors are in this system, the processor with the next lowest designation assumes the processing responsibilities of the secondary processor. We lose jobs scheduled to run on the processor with the

lowest designation because of insufficient remaining computational capacity. The reconfiguration is relatively simple and rapid because copies of the code and the job table reside in the local memories of all processors. The periodic reporting of relevant process state variables in shared memory both serves as a verification that the processes are being executed and as checkpoints in the event that processing must be assumed by a different processor.

When the diagnostic routines indicate that one of the control channels is faulty but the channel can continue to perform at a reduced level, a routine executes to decide which of the available control channels is most appropriate to execute the highest priority processes. This reconfiguration process marks shared memory to indicate which processors are acting with reduced capabilities, and partially reassigns processes in a manner similar to that provided by the job scheduler. The partial reconfiguration option exists because the system may be capable of providing limited control support in the absence of certain sensor or actuator channels. Consequently, we achieve graceful degradation in the case of some multiple faults. In the prototype system, graceful degradation occurs upon detection of a fault in certain specified sensor circuits.

The third possible action occurs on loss of the shared RWM, in which case, no reconfiguration is needed. However, no further reconfiguration is possible because any data sent through the shared memory is suspect. The processors therefore operate independently, and their individual actions depend upon their individual designations. When the primary processor detects the shared RWM failure, it continues operation with a marker in its local memory indicating that it is operating in isolation. Other processors may either shut down or operate in a similar autonomous mode depending on fixed contingency specifications.

System testing

We have thoroughly tested the hardware and software we describe to verify correct operation. In addition to testing the system for correct operation under fault-free conditions, we performed a number of tests involving the insertion of faults. One such test involved starting one of the processors and not the other. In both possible cases, the operational processor recognized that the other processor was not operating and properly assumed control of the system. A second test involved intentionally disabling the transfer of information from the primary processor to the secondary processor. We observed that the secondary processor correctly monitored the lack of information for two interrupt cycles and then assumed control of the system.

We ran additional tests by deliberately injecting hardware faults of various types into the system. For example, one test involved inserting a faulty timer and another involved locking up one of the processors during normal system operation. In all cases, the system correctly detected the existence of the problem and appropriately reconfigured the system.

The work documented here demonstrates a practical approach to implement a fault-tolerant multiprocessor controller. While practical, the approach taken is not necessarily optimal but has been tailored to meet the needs of many real-time control applications, such as the electric wheelchair. The controller offers expanded operational capabilities when compared with a single microprocessor controller, and the system further exhibits self-testing and automatic reconfiguration in response to component faults. The operating system provides the essential functionality for multiple processor scheduling and coordination. Tests performed on the prototype system demonstrate that the system performs as described.

Future work will involve extending the concepts developed in the prototype system to additional applications. For example, we have already extended (although not yet implemented) our approach to an industrial process control application using the Intel 8086 class of processor.[19] Other potential extensions include additional devices for the physically handicapped, such as prostheses, and certain avionics applications. In addition, we continue to investigate techniques for significantly increasing the number of processors employed and possibly interconnecting several of the multiprocessors in a network configuration.

Acknowledgments

The Virginia Center for Innovative Technology, Institute for Computer-aided Engineering, funded our research under a Technology Development Center grant to the Center for Semicustom Integrated Systems at the University of Virginia.

References

1. R.L. Ramey, J.H. Aylor, and R.D. Williams, "Microcomputer-aided Eating for the Severely Handicapped," *Computer*, Vol. 12, No. 1, Jan. 1979, pp. 54-61.
2. R.L. Ramey, J.H. Aylor, and B.W. Johnson, "A Wheelchair-Mounted Manipulator Controller," *Proc. Southeastcon,* Apr. 1980, pp. 272-274.
3. R.L. Ramey, B. W. Johnson, and J.H. Aylor, "Microcomputer-based Aid for the Handicapped Computer Programmer," *Medical and Biological Engineering and Computing,* Vol. 20, No. 5, Sept. 1982, pp. 640-644.

4. J.H. Aylor and B.W. Johnson, "A Microcomputer-based Wheelchair Controller," *Proc. Fourth Ann. Conf. Rehabilitation Eng.,* June 1981, pp. 87-90.

5. B.W. Johnson, "Fault-Tolerant Microprocessor-based Systems," *IEEE Micro,* Vol. 4, No. 6, Dec. 1984, pp. 6-21.

6. B.W. Johnson, *The Design and Analysis of Fault-Tolerant Digital Systems,* Addison-Wesley Publishing Company, Reading, Mass., 1988 (to appear).

7. D. Siewiorek and R. Swarz, *The Theory and Practice of Reliable System Design,* Digital Press, Bedford, Mass., 1982.

8. B.W. Johnson and P.M. Julich, "Fault-Tolerant Computer System for the A129 Helicopter," *IEEE Trans. Aerospace and Electronic Sys.,* Vol. AES-21, No. 2, Mar. 1985, pp. 220-229.

9. L. Philipson and C.W. Heckathorne, "The Use of a Microprocessor as a Versatile Controller for a Powered Wheelchair," *Proc. Fourth Ann. Conf. Rehabilitation Eng.,* June 1981, pp. 84-86.

10. B.W. Johnson and J.H. Aylor, "Design of an Adaptive Controller for Microcomputer Implementation," *IEEE Trans. Industrial Electronics,* Vol. IE-33, No. 1, Feb. 1986, pp. 28-38.

11. J.H. Aylor and K. Short, "A Microprocessor-based Battery Monitor for the Electric Wheelchair," *Proc. Third Int'l Conf. Rehabilitation Eng.,* June 1985, pp. 364-366.

12. J.A. Katzman, "A Fault-Tolerant Computing System," *Proc. 14th Computer Society Int'l Conf.,* Mar. 1977, pp. 77-80.

13. S. Frank and A. Inselberg, "Synapse Tightly Coupled Multiprocessors: A New Approach to Solve Old Problems," *AFIPS Conf. Proc.: Nat'l Computer Conf.,* July 1984, pp. 41-50.

14. B.W. Johnson and J.H. Aylor, "Reliability and Safety Analysis of a Fault-Tolerant Controller," *IEEE Trans. Reliability,* Vol. R-35, No. 4, Oct. 1986, pp. 355-362.

15. T.E. Roberts and B.W. Johnson, "A Fault-Tolerant Multiprocessor for Real-Time Control Applications," *Proc. 1987 Int'l Industrial Electronics Conf.,* Oct. 1987, pp. 488-496.

16. W. Toy and B. Zee, *Computer Hardware/Software Architecture,* Prentice Hall, Englewood Cliffs, N.J., 1986, 446 pp.

17. S. Madnick and J. Donovan, *Operating Systems,* McGraw Hill Publishing Co., New York, 1974, 640 pp.

18. R. Filman and D. Friedman, *Coordinated Computing,* McGraw-Hill, 1984, 370 pp.

19. B.W. Johnson, R.D. Williams, and J.H. Aylor, "A Fault-Tolerant 8086-based Process Controller," Proposal No. EE-ATI-3699-87, Univ. of Virginia, Charlottesville, Apr. 1987.

The biography and picture of **Ronald D. Williams** appear on page 17 in this issue.

Barry W. Johnson is an assistant professor in the department of electrical engineering at the University of Virginia. He is also a member of the Center for Semicustom Integrated Systems, a Technology Development Center of the Virginia Center for Innovative Technology. Previously, he participated in the design and analysis of fault-tolerant computer systems for aerospace applications with Harris Corporation in Melbourne, Florida. His research interests include fault-tolerant computing, VLSI architecture and testing, and microprocessor-based systems.

Johnson received BS, ME, and PhD degrees in electrical engineering from the University of Virginia, Charlottesville. He authored a forthcoming Addison-Wesley textbook, *The Design and Analysis of Fault-Tolerant Digital Systems.* He is a member of the IEEE, the Computer Society, Tau Beta Pi, Eta Kappa Nu, and Sigma Xi. He actively participates in the Computer Society as treasurer and as a member of the Executive Committee, the Membership and Information Board, and the editorial board of *IEEE Micro.*

Thomas E. Roberts is an instrumentation and control systems engineer for the US Navy. He received the BS and MS degrees in electrical engineering from the University of Virginia, where he researched the development and analysis of fault-tolerant, microprocessor-based systems.

Roberts is a member of the IEEE, Tau Beta Pi, and Eta Kappa Nu.

Questions concerning this article can be addressed to Barry W. Johnson, Dept. of Electrical Engineering, University of Virginia, Charlottesville, VA 22901.

FEATURE

Software Fault Tolerance in Architectures with Hierarchical Protection Levels

Here's how to implement fault-tolerant software in a popular microprocessor.

Brenda M. Ozaki
IBM Corporation

Eduardo B. Fernandez
Florida Atlantic University

Ehud Gudes
Ben Gurion University

An increasing number of computer applications require a high level of reliability. One way of reaching this reliability level is by using fault tolerance. Fault tolerance is the ability of a system to perform according to its specifications in the presence of errors. Although hardware fault tolerance has been studied in great detail, software fault tolerance has not received the same amount of scrutiny. In fact, a dearth of studies addresses how specific architectural hardware features can support software fault tolerance. As such, this article is a contribution to realization studies of fault tolerance. While many theoretical studies have been published on this subject, few analyses of their implementation in specific environments exist.

Here, we consider the effect on software fault tolerance of hardware features such as hierarchical privilege levels (rings), the use of descriptors for memory protection, separated virtual address spaces, and ring crossings that enforce specific entry points. These features have proved to be effective for security.[1] Several commercial and experimental architectures use similar features (among them, Multics, where most of these concepts were introduced).[2] These architectures include the Prime 50, the Data General MV/8000[3] and MV/10000, and the Intel 80286[4] and 80386.[5] For the sake of concreteness, we selected the Intel 80286 as a sample implementation vehicle. However, most of our discussion applies to any machine with a similar range of features.

The key to achieving fault tolerance is redundancy. When applied to software, redundancy may appear as either

- N-version programming, or
- recovery blocks.

N-version programming involves the execution of several independently developed versions of a software algorithm and a voting system that outputs the majority result.[6] In the recovery-block approach, the state of the computation is saved in a recovery point before execution of a block. At completion of the block, an acceptance test is applied. If the test is successful, the computation proceeds to the following block. If not, the recovery point is used to restart the failed computation, and an alternative block then executes. This process continues until the acceptance test succeeds or the complete computation fails.[7] (See accompanying box on software fault tolerance.)

The N-version programming and recovery-block approaches are general and represent different aspects of design diversity. They can be applied to

0272-1732/88/0800-0030$01.00 © 1988 IEEE

Software Fault Tolerance

Software failures come from design errors. Regardless of the effort spent in testing and debugging complex software, some errors remain. These errors could have a catastrophic effect on the operation of real-time systems. The consequent need to mask the effect of design errors is the objective of software fault-tolerant mechanisms. The most commonly used approaches are N-version programming and recovery blocks.

N-version (or multiversion) programming implies the development of N (± 2) independent versions of the same program (using the same specifications).[1] *Independent* means that the approaches to implementation should differ as much as possible and even that different programmers should be used to avoid correlated errors.

Different approaches can be used depending on the value of N. Clearly, if $N = 2$, we can only perform a comparison for error detection at the end of the computation or after determining specific values along the program. Again, for $N \pm 3$, voting can be used only at the end of the computation or at predefined points where specific variables are determined.

N-version programming is the subject of several studies.[1,2] In particular, Knight, Levenson, and St. Jean describe a study that indicates a correlation between the errors committed by teams of independent programmers. Eckhardt and Lee have analyzed the effect of these correlated errors on reliability.[3] Their conclusion is that N-version programming is still a better strategy for reducing the probability of system failure than single-version programming.

Two-version software has appeared in the US space shuttle, the Airbus A-310 secondary flight-control system, the Boeing 737-300, and the Boeing 757-767 yaw damper system. The Bendix flight-control system uses three-version software.[4]

A group at the University of Newcastle in England developed the concept of *recovery block*. The syntactic form of recovery block (RB) is:

RB: *ensure* AT
by B_1
else by B_2
...
else by B_n
else ERROR

AT stands for *acceptance test*, that is, a logical expression or condition that must be true for these results to be acceptable. For example, a result must be positive, its inverse must be equal to some input value, and a continuity condition must hold. B_1 is the *primary block* (module) and B_2 to B_n are *alternate blocks*. B_1 to B_n represent different ways of performing a particular calculation. The state of the process in entering the calculation is saved at that point, called a *recovery point*. AT is evaluated after executing B_1. If successful, the computation exits the block. If not, the initial state is restored and an alternate block is executed. If AT keeps failing, we repeat this process until we run out of alternate blocks. Then we either give up or resort to other measures such as reconfiguration with degraded performance.

any type of computation, sequential or concurrent. For concurrent systems, the main concern is to avoid the so-called *domino effect*, where the failure of an acceptance test provokes an uncontrolled series of process rollbacks.[7] Specifically, when these mechanisms are applied to concurrent systems, they take the form of

- programmer-transparent coordination, a technique in which the operating system adds recovery points at specific points[8] (these additional recovery points limit rollback propagation); or
- conversation, the equivalent to a concurrent recovery block.[7]

Coordination among concurrent processes is performed by recovery points and acceptance tests for each process. All processes must pass their acceptance tests, or the conversation fails and an alternative set of processes is invoked. The participating processes must not communicate with any other processes while engaged in the conversation, that is, the equivalent of *sidewalls* must be enforced. The combination of the recovery line (set of recovery points), test line (set of acceptance tests), and the sidewalls defines an *atomic action*.

The use of a strategy that employs a separate programming layer, the *recovery layer*, to handle fault-tolerant aspects of process interaction has been proposed in several studies.[9-11] A *recovery metaprogram* (RMP)—which monitors the run-time behavior of the application program and coordinates error detection, recovery, and reconfiguration—executes in this recovery layer. This architecture was shown to be a convenient way of unifying different ad hoc recovery mechanisms, and as such it constitutes a good framework to evaluate recovery implementations. A set of recovery layer primitives that can be used to define different fault

Recovery blocks can be nested. Also, the alternate blocks may provide more detailed—or less detailed—ways of performing the calculation. For example, a calculation that results in an overflow could be performed by using a double-precision algorithm. In such a case, AT has to take the different precisions into account.

A possible difficulty with this method is the definition of a good acceptance test. Sometimes this definition is easy, as in functions with a reasonably calculated inverse or in algorithms such as sort. In most cases, however, the quality of the error coverage provided by a given AT is unclear.

Neither method possesses inherent superiority. The N-version approach requires N processors to be time efficient because all versions must execute to vote on their outputs. On the other hand, in the recovery block approach the alternate blocks do not execute unless the primary block fails. Again, finding a good AT may not be easy—voting is inherently simpler.

Concurrent software now uses only methods based on recovery blocks. Examples of this approach are the Conversation and the Programmer-Transparent Coordination.[5,6]

References

1. A. Avizienis and J.P.J. Kelly, "Fault Tolerance by Design Diversity: Concepts and Experiments," *Computer*, Vol. 17, No. 8, Aug. 1984, pp. 67-80.
2. J.C. Knight, N. Levenson, and L.D. St. Jean, "A Large Scale Experiment in N-version Programming," *Proc. 15th Ann. Intl Symp. on Fault-Tolerant Computers*, CS Press, Los Alamitos, Calif., 1985, pp. 135-139.
3. D.E. Eckhardt, Jr. and L.D. Lee, "A Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Error," *IEEE Trans. Software Eng.*, Vol. SE-11, No. 12, Dec. 1985, pp. 1511-1517.
4. D.P. Gluch and M.J. Paul, "Fault Tolerance in Distributed Digital Fly-by-Wire Flight Control Systems," *Proc. IEEE/AIAA Seventh Digital Avionics Systems Conf.*, 1986.
5. K.H. Kim, "An Approach to Programmer-transparent Coordination of Recovering Parallel Processes and Its Efficient Implementation Rules," *Proc. 1978 Conf. Parallel Processing*, CS Press, Los Alamitos, Calif., Aug. 1978, pp. 58-68.
6. K.H. Kim, "Software Fault Tolerance," in *Handbook of Software Engineering*, C.R. Vick and C.V. Ramamoorthy, eds., Van Nostrand Reinhold Co., New York, chap. 20.

tolerance strategies is proposed in Fernandez et al.[11] The RMP itself must employ lower level primitives to implement the primitives used in the recovery level. In this sense, the implementation we discuss could possibly support these recovery primitives and take advantage of a specific machine-level architecture. If the recovery layer included primitives that use explicitly different environments of execution,[9] then the architecture described here would be a direct implementation of those directives.

We discuss the recovery layer with emphasis on how the protection structure of the 80286 supports its implementation. Since we need to consider the effect of a kernel, we use Intel's extension kernel, the K286, as a representative set of kernel primitives.[12] We cover additional implementations of software fault-tolerant methods as they may be adapted to the 80286:

- recovery blocks,
- conversations, and
- time-outs.

All of these methods are implemented using an RMP. We also consider the effect of multiprocessing in this implementation.

Software fault tolerance can be implemented without the use of an RMP. Obviously, many alternatives exist. We chose our approach because it is general and flexible and makes good use of the specific features of the 80286/80386 architectures. We show here that the use of privilege levels and descriptors can support a distribution of functions that contributes to the total reliability of the executing software. Hamer Hodges, among others, showed that capabilities (a refinement of the concepts of descriptors and rings) are valuable for fault tolerance.[13] Here we show that a simpler structure than that of capabilities is still effective.

We discuss the software architecture of the kernel, the RMP, and the application program. We consider how descriptors and rings affect reliability. Then we describe how the RMP can be used to support fault-tolerance concepts.

RMP and privilege levels

A fault-tolerant architecture using a separate recovery layer has been the subject of previous studies.[9-11] This recovery layer includes the structure of the fault-tolerance and recovery functions. This layer is the highest level abstraction of the system. It has a global view of the system resources that thus can be monitored and reconfigured for recovery actions. The specific embodiment of a set of these functions is the

RMP. The type of system dictates whether one or several RMPs are employed. As previously discussed, the RMP monitors the run-time behavior of the application program and coordinates the error detection, recovery, and reconfiguration actions for the system. This separation provides recovery transparency for the application program and protects the recovery functions themselves from errors at the application level. To enforce this protection, the hardware architecture must provide ways to isolate the RMP.

The recovery metaprogram. The RMP consists of

- a set of procedures implementing the algorithms selected to accomplish application-independent, fault-tolerant actions (for example, recovery blocks or conversations);
- the acceptance tests, voting, and other evaluation and monitoring actions defined by the application program; and
- a set of procedures that can decide on reconfiguration strategies in case of failures (actual reconfiguration is performed by the kernel).

These elements of the RMP are implemented by means of suitable primitives, some of which we define later.[11] Here, we map this recovery architecture into an 80286 architecture. To provide an understanding of this mapping, we first review some of the relevant features of the 80286. (For more details, consult Childs et al.,[4] Intel publications,[5,12,14,15] and Morse and Albert.[16])

The protected mode of the 80286 uses selectors and descriptors that implement a virtual address space.[4,16] The descriptors perform address translation and memory access control. Figure 1 shows the structure of a descriptor. The limit and the base define the boundaries of a segment, while the access rights indicate the allowed use of this segment. All the descriptors needed by a task are set up in a local descriptor table. Separation of tasks is accomplished by giving them different LDTs, since these tables represent the task access rights. The 80286 also provides four hierarchical privilege levels or rings that may be used to protect and separate programs and data. Controlled transfers from one state to another are performed by means of gates (Figure 2), which are special descriptors that enforce protected entry points where parameter validation can be performed.

The most trusted and most privileged level is level zero. Subsequent levels are less trusted and are labeled as levels one, two, and three. By using different privilege levels for the RMP and the application program, the 80286 provides separate execution environments for the programs. Since we also need a kernel, we can make use of the architecture of the 80286 to define three separate layers to support these concepts: the kernel, recovery, and application layers.

The kernel layer controls the memory management, scheduling, and multitasking operations for the system

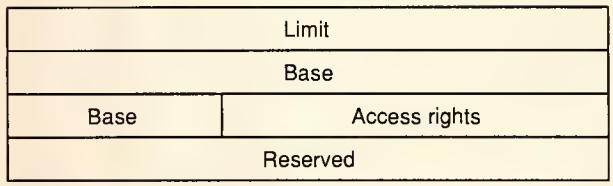


Figure 1. Descriptor format.

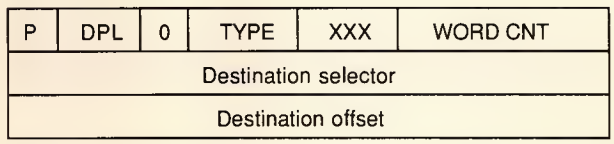


Figure 2. Gate descriptor. P is the present bit, DPL is the descriptor privilege level, TYPE selects the type of gate, and WORD CNT is the number of words copied from the caller's stack to the called procedure's stack (Call Gate Only). The selector and offset specify an entry point at the destination ring.

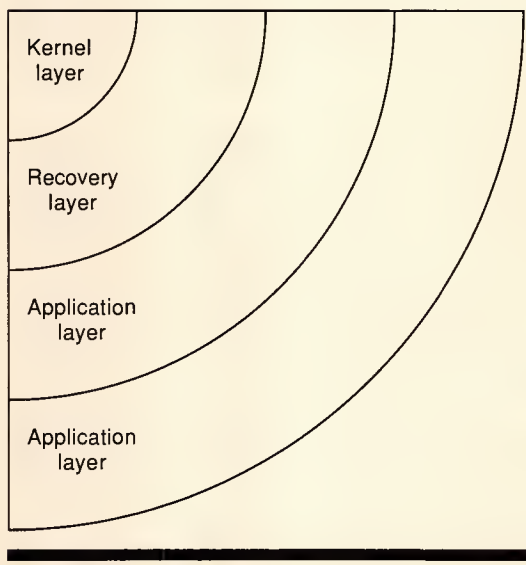


Figure 3. Software layers.

and resides at level zero. It handles the traps and most of the interrupts for the system. The RMP resides at level one. It handles specific system-defined interrupts such as software interrupts, as well as error recovery for the application programs. Applications may reside in levels two and three (Figure 3). Separating the kernel, the RMP, and the application programs into different privilege levels provides good protection against error propagation. It also protects each component against errors in the other components. This separation is also useful to enforce security restrictions, since possible at-

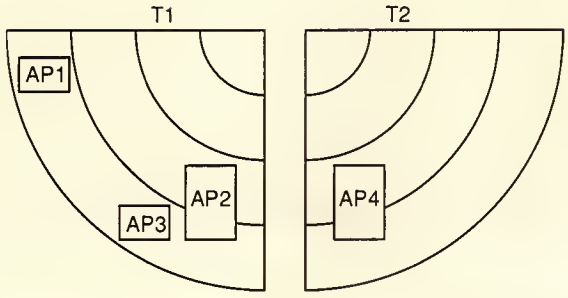


Figure 4. Tasks and applications. Task T1 supports three applications that span two privilege levels. Task T2 supports one application that spans two privilege levels.

tacks could take advantage of unchecked system calls or unchecked parameters.[1] This protection is based on the inherent isolation provided by separate address spaces and enforced by the descriptor and ring structure. Descriptors also prevent errors due to misuse of data types since their access types can enforce their correct use. For example, since a procedure is execute-only, it cannot be modified by an erroneous program. We describe each of these layers in detail to show how this protection is accomplished.

Application layer. For a multitasking environment, tasks may be used to run application programs. Tasks in an 80286 are protected from one another because separate descriptor tables are used for each task, which defines separate address spaces. This ability helps contain faults within the task in which they occur and prevents their propagation into another task. An application program may be created as a single task or as a set of tasks, and it may use several layers of privilege, depending upon the access needs of the code. Dividing an application program into different tasks or privilege levels provides controlled accesses within an application (Figure 4). For example, untested programs could run in the least privileged level.

The choice between structuring an application as a set of tasks or as a single task depends on several factors. For maximum performance, a single task is preferable since the overhead of task switching is considerable (over 200 cycles in the 80286, close to 300 in the 80386). For maximum reliability, the use of tasks adds another barrier to error propagation because of the separate sets of descriptors.

Recovery layer. Since the RMP resides at level one while the application program resides at level two or three, the application program cannot call the RMP directly but must use a gate to access its services. An RMP can be seen as a coroutine with respect to the application program or as a separate task. The application calls the RMP using either a call gate or a task gate.

If the application uses a call gate, it can pass parameters to the RMP by pushing them on its stack and indicating their number to the called procedure in the word count field. The RMP checks any selectors that the application has passed it to ensure that the application has sufficient privilege to send the selectors, that is, that they are valid parameters. The following instructions[14] can be used by the RMP to validate parameters:

- Load Segment Limit,
- Load Access Rights Byte,
- Verify Segment Readable, and
- Verify Segment Writable.

For example, the LSL instruction specifies a selector and a register. If the descriptor denoted by the selector is visible at the current privilege level (the executing program has sufficient privilege to access the descriptor), then the limit is loaded into the specified register. The limit can then be compared to the offset that the program uses. If the offset is within the limit, the access is valid; if not, a trap occurs when the access is attempted. The other instructions perform similar functions. The use of these instructions prevents a significant range of errors.

If the application program uses a task gate to access the RMP, it needs a mechanism such as a mailbox for communication with the RMP. The K286 provides several mailbox primitives, some of which are described later.

In the case of multiprocessing, the recovery layer should also monitor hardware functions, especially those intended for fault detection. It must also be aware of which software components could be affected by hardware malfunctions. Hence, the components and topology of the system must be identifiable and controllable from within the recovery language.[9] Although the resolution of this problem is beyond the scope of this article, these issues illustrate the need for the RMP to monitor hardware faults.

The RMP could have the ability to run a set of diagnostics on the system at power up—or at any time when an error is detected—to determine whether the problem was a transient error or the result of a system component failure. If the RMP determines that a system component is faulty, it notifies the kernel so that the system can be reconfigured.

Kernel layer. In addition to its normal functions of task scheduling and memory management, the kernel handles the traps and most of the interrupts for the system. The kernel uses the error code that is pushed on the stack for a trap as the kernel attempts to recover from the trap. Interrupts and traps may be handled by an interrupt/trap gate or a task gate.

The RMP needs the services of the kernel to roll back an application program. Several privileged instructions are necessary to restore the system address registers of the 80286. The registers may be stored by a program executing at any privilege level but can only be loaded by a program executing at level zero. For example, the RMP can save the contents of the LDT address register (LDTR) but cannot load the LDTR. Only the kernel at level zero can load the LDTR. This situation presents a potential reliability problem since an untrusted program can save the segment registers.[17] This problem could be solved by trapping any segment register writes to the kernel or by only allowing the segment registers to be loaded from a level zero data segment.

The 80286 uses a Task State Segment to support task switching. The TSS includes the current contents for all data and control registers. Using the TSS may improve performance for checkpointing, since the TSS includes information that will be needed to restore the execution environment of a failed block execution.

Multitasking considerations. Multitasking in a single processor requires the coordination of several applications with the recovery layer. For RMPs that operate as special subroutines of the application programs, little difference exists between a system running a single task or a multitasking system. In this case, each application would have its own RMP. A multitasking system could use a single RMP that would be in a different task from any of the application tasks. Additionally, RMPs could be associated with specific tasks or applications, depending upon the recovery needs of the processes. System-level multitasking issues—resource management and task scheduling—are managed by the kernel. Implementing these programs as separate tasks provides more flexibility, since they could be assigned to different processors within a multiprocessor cluster. However, this practice would incur extra overhead due to the need to store TSSs at task switching.

For an RMP that operates as a separate task from the application programs, task coordination and prioritization are key issues. Since task communication is handled as a Remote Procedure Call (RPC) in most concurrent languages, the RMP and the application program must be synchronized to communicate. In K286 mailbox primitives, if one task issues a send before the other task issues a receive, the first task is suspended until the receive is issued. Once the message is received, each task can resume execution. The RMP should be a high-priority task to be responsive to an application's needs.

For intertask communication, some type of mailbox must be defined in which tasks can pass data and/or descriptors to one another. As indicated earlier, Intel's extension kernel, the K286, provides several mailbox primitives that could be used for this purpose.[12] Mailboxes exist at privilege level zero and are defined by the KQ_CREATE_MAILBOX primitive. A send privilege level (SPL) and a usage privilege level (UPL) are specified when the mailbox is created. A task must have both sufficient privilege and a descriptor (for the mailbox) to use or alter the mailbox in any way. The SPL specifies the minimum privilege level that a task must have to delete the mailbox or to send it as part of a message. That is, an alias for the mailbox descriptor may be sent to another task, giving it access to the mailbox. The UPL specifies the minimum privilege level that a task must have to send or receive messages via the mailbox. Mailboxes are accessed from lower privilege levels through gates. Other mailbox primitives include KQ_SEND, KQ_RECEIVE, and KQ_QUERY. KQ_SEND and KQ_RECEIVE are used to send and receive messages via a mailbox. KQ_QUERY allows a task to determine the status of a particular mailbox. The application program and the RMP exchange messages consisting of data and/or descriptors via their shared mailbox. Since a K286 mailbox resides at level zero, the application needs to use a gate to access the mailbox.

Privilege levels provide protection against error propagation.

If priority scheduling is desired, then the kernel sets priority levels, time-slice intervals, and/or time limits for each task using the primitives of the K286. This procedure allows high-priority applications to obtain better service from the RMP.

RMP implementation

Here we consider several fault-tolerance mechanisms with respect to how they can be supported by the RMP in the context of the 80286 architecture. We do not discuss the details of the RMP itself, but we do show how the generic actions needed by this program to support fault tolerance can be implemented in this environment.

Recovery blocks. In an 80286 architecture, a recovery block can be implemented by passing the selectors for the Primary Try, Alternative Tries, and Acceptance Test to the RMP. The RMP then coor-

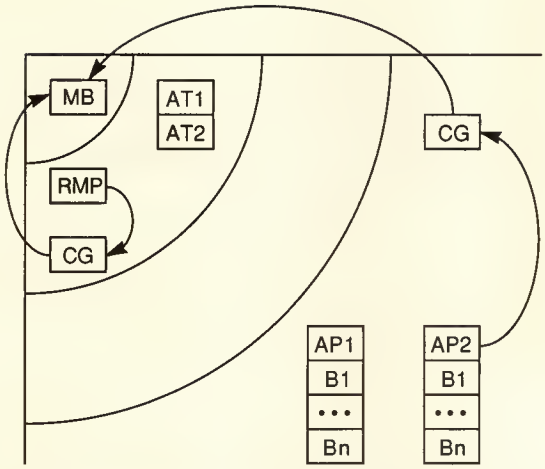


Figure 5. Recovery block, single processor. APns are the application programs, Bns are the try blocks, ATs are the acceptance tests, MB is a mailbox used by the applications and the RMP to communicate, and CG is the call gate to the mailbox. The arrows detail one call from the application program to the RMP using a mailbox.

dinates the execution of the recovery block (Figure 5). The compiler defines the Primary Try, Secondary Try, and Acceptance Test as procedures, and the selectors for these procedures are passed to the RMP. The RMP initiates the execution of the Primary Try and applies the Acceptance Test. If it passes it, the RMP returns to the application without initiating the Secondary Try. If the Primary Try fails the Acceptance Test, then the RMP initiates another try until no more tries are left. At this point, the RMP executes an error-handling routine or an abort procedure. Since recovery blocks correspond to logical program units (subroutines, modules, blocks), the compiler inserts a call to the RMP to save the variables that will be changed by the try block program and allocates storage for local variables. If the Try Block fails the Acceptance Test, the variables are restored and the next Try Block is executed.

A recovery block can be described in terms of the primitives defined in Fernandez et al.[11] as shown in Figure 6.

In the 80286, calls from the application to the RMP are implemented as state transfers that use gates by means of a call instruction. Since no calls are allowed from a higher privilege to a lower privilege ring, the RMP must push the starting address of the block procedures—B—in its stack and execute a return instruction. In effect, this implements the RMP and the executing block as coroutines. In Figure 7, we show the details of these steps. They can be considered as the lower level implementation of the RMP primitives for the particular environment considered here.

The pattern in Figure 7 continues until a Try Block passes the Acceptance Test or no more Try Blocks exist. Whenever a new Try Block is initiated, the values that the previous Try Block changed are restored to their original values using the rollback feature of the RMP. If the last Try Block fails, the RMP uses the Fail procedure specific to the application.

Each call to the RMP from the application program requires a call gate descriptor (CG). If the call is between tasks, a mailbox is required. When an application uses a call gate only, the Try Block result is passed to the RMP by pushing the result onto the stack. This result is then copied to the stack of the RMP. The word-count field of the call gate specifies how many words are to be copied from the level three stack to the level one stack. If the application uses a mailbox, the result may be sent as part of the message to the RMP.

```
process RB;
begin
SET__UP(RB)        —initializes execution environment
at RB__entry
    do
       loop i = 1 to N                          —there are N blocks
          SAVE__ENV                             —saves global variables
          ACTIVATE(Bi)                          —activates try block
          if VALID__RESULT(Bi) then exit        —acceptance test evaluation
          else RESTORE__ENV(Bi)                 —restores global variables
       end loop
       if i > N then FAIL
       else DISCARD__ENV                        —discard execution environment

    od
end;
```

Figure 6. Description of a recovery block (RB) in terms of primitives.

```
Application: RMP

RB: CALL RMP(AT,Bi,SD)
      —Call RMP passing it the
       descriptors for the acceptance
       test (AT), recovery block (Bi),
       and the data, SD.

                                    INITIALIZE (Bi)
                                      —Corresponds to SET_UP primitive
                                      KQ_CREATE_SEGMENT (ENV)
                                      —allocates storage for saving
                                      environment and creates a descriptor
                                      for this segment.
                                    PUSH Bi
                                      —saves start address of Bi into
                                      —level one stack.
                                    RETURN
                                      —activates Bi

      —Execute whatever function
      has to be performed
      CALL RMP_Test(Bi result)
      —Send result to RMP for validation

                                      —Apply validation test
                                    IF PASS THEN
                                      BEGIN
                                      KQ_DELETE_SEGMENT(ENV)
                                      RETURN
                                      END
                                    ELSE
                                      —restore environment and try next
                                      block
```

Figure 7. Detailed steps of a recovery block using an RMP.

Conversations

Processes involved in a conversation would be tasks in an 80286 environment. Since each task has a valid address space that is specified by its LDT and its privilege level, sidewalls can be enforced by the use of appropriate descriptors. In a conversation, the RMP saves data at the recovery points, performs the local and global acceptance tests, and manages the rollback if necessary. The RMP is aware of all interprocess communication (Figure 8). The functions to be performed by each participant component in the conversation are:

- *Recovery point*. The RMP checkpoints all processes (tasks) at the recovery line. Each process accesses the RMP through a gate.
- *Process execution*. Processes execute in parallel and communicate through a monitor or by direct message-passing.
- *Local acceptance tests* (LATs). The RMP performs LATs for each process.

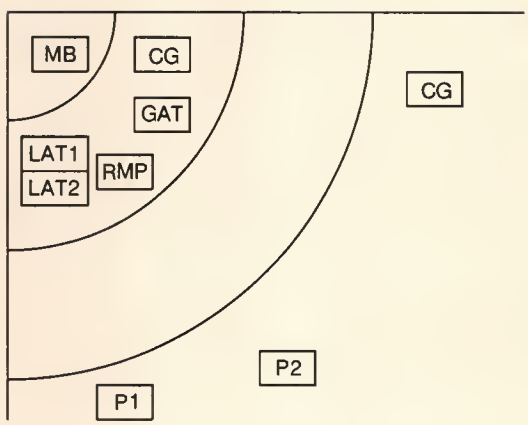


Figure 8. Conversation, single processor. Pns are the processes in the conversation, LATns are the local acceptance tests, and GAT is the global acceptance test.

Process n:

JOIN_CONVERSATION

- create child task (KQ_CREATE_LDT, KQ_CREATE_TASK).
- copy LDT to child task (KQ_MOVE_TO_NEW_LDT, KQ_NEW_LDT_GATE).
- suspend child task (KQ_SUSPEND).
- create aliases for LAT and GAT (KQ_CREATE_ALIAS).
- send message to RMP with the id of the parent and child tasks and the descriptors for the LAT and the GAT (KQ_SEND).

Recovery Metaprogram:

INIT_CONVERSATION

- if this is the first process in the conversation then create a CCL and name it (KQ_CREATE_SEGMENT).

START_PROCESS

- add PCB for the process to the CCL (KQ_GROW_SEGMENT).

Processes:

- Processes communicate via mailboxes (KQ_SEND, KQ_RECEIVE).

Process n:

CALL RMP_test(Process_n _result)

QUIT_CONVERSATION

- send message to RMP with the id of the parent task (KQ_SEND).

Recovery Metaprogram:

END_PROCESS

- find LAT descriptor for Pn in its PCB and apply acceptance test.
- if LAT passes, then set PCB for Pn to 'complete.'
- if LAT fails, then call ROLL_BACK.

END_CONVERSATION

- if this is the last process then find descriptor for GAT and apply acceptance test.
- if GAT passes then delete CCL and return (KQ_DELETE_SEGMENT).
- if GAT fails, then call ROLL_BACK.

ROLL_BACK

- find parent task ids in the PCBs and delete parent tasks (KQ_END_TASK, KQ_DELETE_TASK).
- find child task ids in the PCBs and initiate child tasks (KQ_RESUME).

Figure 9. Detailed steps of a conversation using an RMP.

• *Global acceptance test* (GAT). The RMP performs a GAT on each process at the acceptance line. If any process fails, all processes are rolled back to the recovery line. If all the processes pass, the recovery line is discarded and execution continues.

We examine in detail the primitives that are needed to implement a conversation using an RMP. We define these primitives in terms of K286 primitives. Several primitives exist that provide support for task management and scheduling in a multitasking environment. Task origination results from first creating a new LDT by using the KQ_CREATE_LDT primitive. Then the KQ_CREATE_TASK primitive may be used to create a new task and assign it initial scheduling parameters and a task identification. A task can copy its LDT to a newly created task by using the KQ_MOVE_TO_NEW_LDT and KQ_NEW_LDT_GATE primitives. Tasks can be temporarily or permanently suspended. A KQ_SUSPEND is used to halt the execution of a task until a KQ_RESUME primitive is issued, a KQ_END_TASK ends a task permanently, and a KQ_DELETE_TASK deletes the LDT of a task that was ended by the KQ_END_TASK. K286 memory management primitives support segment creation and sharing. A KQ_CREATE_SEGMENT produces a new segment by creating a descriptor and assigning a selector to it. The limit of a segment can be increased by using the KQ_GROW_SEGMENT primitive. An alias can be created for a segment by using the KQ_CREATE_ALIAS primitive. The alias may have different access rights than the original segment did.

Some of the details depend on the specific mechanisms used for the conversation.[18] Here, we consider a generic or typical conversation structure. For this example, the RMP may be seen as an endless loop that waits for messages. For each conversation, the RMP creates a conversation control list. Each process in the conversation is represented on the CCL by a process control block (PCB) that contains the ids of the parent and child tasks and the acceptance test descriptors. The child task is created to save the process state and is used if a rollback is necessary (see Figure 9). This conversation can also be expressed first in terms of RMP primitives, but we avoid doing this to preserve simplicity. That kind of description can be found elsewhere.[11]

The descriptors needed to implement a conversation using an RMP in an 80286 environment are

• *RMP*: LAT1, LAT2,..., LATn, GAT, CCL, CG (to mailbox), CG (to kernel).
• *Process n*: LATn, GAT, CG (to mailbox), CG (to kernel).

If the processes are written in Ada or another language that supports exceptions, the processes would need additional descriptors for the exception handlers. If an exception was handled at the recovery layer as a software interrupt, the descriptor would be a trap/interrupt gate. If an exception is handled at the application layer, then a selector for a gate could be used by the exception handler to call the RMP for recovery services.

The RMP saves data at the recovery point for the conversation.

Time-outs. In this example, one processor executes two processes, T1 and T2. Process T1 requests a service from T2 by invoking one of its procedures, T2_Proc, which passes some results as return parameters of the call. We detail the situation of the RMP when it imposes a time-out on the answer from T2_Proc. This case is based on an example found in Ancona et al.[9]

The two processes, T1 and T2, are treated as tasks in the 80286 environment. Therefore, T2_Proc is a code segment defined within task T2. If the call for T2_Proc is completed successfully, T2 passes return parameters to T1. Assume that the processes T1 and T2 are applications running at privilege level three and that the RMP runs at privilege level one.

Two mailboxes are needed, MB1 and MB2. MB1 is used to pass messages between T1 and the RMP, and MB2 is used to pass messages between T2 and the RMP. All mailboxes reside at privilege level zero and are accessed through call gates from lower privilege levels as defined by the K286.

T1 puts the time-out value and the selector for T2_Proc into a mailbox that it shares with RMP. Once RMP1 receives the message, it checkpoints the current state of T1 and sets an external timer that provides a hardware interrupt when a response is not received from T2_Proc in the specified amount of time. If the time-out occurs, the interrupt is serviced by an interrupt handler selected from the interrupt descriptor table. The RMP calls the kernel and requests that it activate T2_Proc. If T2_Proc is completed successfully, then T2 passes the results to RMP via the mailbox. The results are then sent to T1. The RMP resets the timer once it receives a message from T2. When T1 receives the data from MB1 it continues executing.

See Table 1 for a conceptual description of the interaction of the processes. Using K286 primitives would implement this interaction similarly to the other two cases. If the time-out interrupt is received, the interrupt handler calls the scheduler in the kernel and requests that T2 be cancelled and that control be returned to the RMP. A flag of some type must also be passed to the RMP to indicate that a time-out has occurred. At this point the RMP could try to take some action such as

Table 1.
Conceptual description of process interaction.

Active process	Operation
T1	Send message to RMP which includes selector for T2-Proc and timeout value.
RMP	Receive message from T1. Checkpoint T1. Set timer. Send message to T2 which includes selector for T2__Proc. Call kernel.
Kernel	Activate T2.
T2	Receive message from RMP. Execute T2__Proc. Send results to RMP.
RMP	Receive results from T2. Resets timer. Send results to T1. Call kernel.
Kernel	Activate T1.
T1	Receive results from RMP. Continue executing.

Table 2.
Time-out sequence.

Active process	Operation
T2	Receive message from RMP. Execute T2__Proc.
Timeout	Error occurs, T2 does not return result of T2__Proc.
Interrupt handler	Call kernel to abort T2 and activate RMP.
RMP	Send T1 message with error indicator. Possible to rollback and retry call.
Kernel	Activate RMP.
T1	Receive message.

retrying the request, or it could simply return a message to T1 indicating that the requested service was not performed. See Table 2 for the time-out sequence that begins with the activation of T2. Distinctions between time-outs and normal returns can be accomplished in this architecture by taking advantage of the separate CALL/INTERRUPT and RETURN/IRETURN instructions.

Multiprocessor systems

A multiprocessor system may be used to provide fault tolerance through redundancy or performance improvements through parallelism. The software fault-tolerant constructs that we discussed for single-processor systems may be extended to multiprocessor systems.

We assume that a multiprocessor system has some shared physical and local memory and that any processor can communicate with any other processor. We consider here an environment where tasks are statically assigned to processors unless otherwise stated. The virtual address space for a multiprocessor system could use one Global Descriptor Table (GDT) for all processors and multiple LDTs per processor. LDTs would not be shared between processors except for backup purposes.

Recovery layer. In a multiprocessor system that uses the recovery-layer approach, each processor supports a recovery layer. The recovery layer may be a single RMP or multiple RMPs depending upon the needs of the processes it supports. The RMPs may differ from processor to processor, but all must provide basic recovery services. The RMPs would be similar to those previously described but would require some additional features. All interprocessor communication would be visible to the RMP and could use a timer, set by the RMP, to ensure that recovery actions could be taken if a processor does not respond within a prescribed amount of time. Prior to the interprocessor accesses, the RMP would checkpoint the current status to use if recovery action was necessary.

RMP coordination is an important issue. If recovery structures are distributed, a supervisory RMP must coordinate the RMPs in the distributed processors. Each RMP should be capable of supervising the recovery actions required by the applications it is running. Backups should be maintained for all supervisory programs. Reconfiguration then applies to the supervisory program as well as to the processors.

Reconfiguration requests could come from the recovery layer as part of a reconfigure-then-retry approach to error recovery, or from the detection of an error through diagnostics. If the RMP keeps an error log, multiple failures of the acceptance test in a recovery block or conversation could trigger a reconfiguration request to disable the faulty processor.

Reconfiguration requests and decisions are handled at the recovery layer, while actual resource reconfiguration is handled at the kernel level.

Conversation. In a multiprocessor system, each process could run in a separate processor with its own RMP. The processes are synchronized by the use of RPCs. Each RMP checkpoints the process it is running to establish a recovery line. As the processes require,

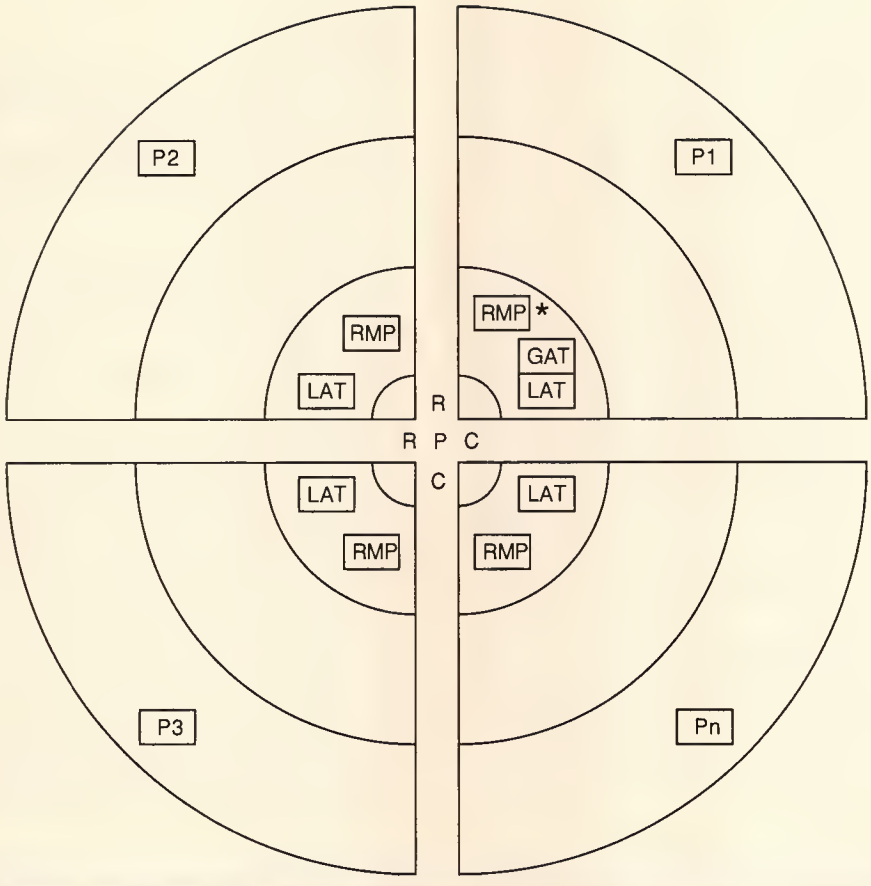


Figure 10. Conversation, multiprocessor case. Assume the starred RMP is the supervisor.

the RMPs perform LATs. The GAT is performed by the RMP of the supervising processor for the conversation. If the GAT fails, all processors are instructed to roll back their processes to the recovery line and try again (Figure 10).

Conclusions

The 80286 and the software layers have been shown to support conveniently the methodologies for fault-tolerant software described earlier. Existing software fault-tolerant methods—recovery blocks, exceptions, N-version programming, conversations, and time-outs—are supported by the recovery-layer approach, and we have shown here that there is a convenient mapping of the recovery layer concepts to the implementation entities.

Use of the RMP will affect performance as it introduces another level of indirection for some system calls. For example, for the N-version case described earlier, if we assume N = 3, we would need a maximum of 13 calls to the RMP, one for SET_UP, three to save environments, three to activate processes, three to validate results, two to restore environments, and one to discard the environment (this assumes that the third version passed the acceptance test). An 80286 running at 10 MHz can perform a call to a segment at a higher privilege level in 8.2 microseconds and a task switch in 18 μs. Without the support of the 80286 microcode, a task switch would be two to three times slower than these values.[15] If we use these values, the overhead is 13 μs $\times$ 8.2 μs = 106.6 μs for the execution of a three-version module implemented as a set of coroutines and 13 μs $\times$ 18 μs = 234 μs when using separate tasks. Note that this scenario is worst case. The first try block usually succeeds, so the average overhead is considerably less. The percentage of overhead depends on the application and cannot be determined in general. This overhead is clearly the price that has to be paid to provide a high level of reliability and security. An architecture not providing these features would not incur in this overhead but, as indicated earlier, the RMP does not protect against errors or attacks coming from the application programs. Since these errors or attacks come from the dynamic interaction of the processes at exe-

cution time, there is no way to provide protection either by compile-time analysis or by additional protection software. In other words, an architecture not providing these—or equivalent—features would be considerably more vulnerable to these types of problems.

As indicated earlier, the operations described here can be seen as a lower level implementation of recovery primitives. The present case would be one of many possibilities to implement those primitives. Another possibility is that those primitives include specification of execution environments using statements such as Execute and others, as proposed in McKendry and Campbell.[19] In that case, one can specify at the RMP level where the RMP and the application programs should execute and when a change of environment is needed. The parameters in the service calls of the Execute statement are protected using capabilities; in an architecture of the type considered here these parameters would be protected using descriptors.[19] While capabilities are considered a stronger protecting mechanism, their efficient implementation is still an open problem.

While this discussion has used a specific system, the Intel 80286, these results are valid for a range of similar systems. Several architectures use rings, starting from their first implementation in Multics to the Prime 50, and others. Very specifically, the Intel 80386 has basically the same architecture as the 80286 and can provide similar support.

Because the kernel defines a virtual machine on top of the hardware architecture, its effect is also very important. When one applies this approach to those other machines, a set of primitives similar to those provided by the K286 should be available.

As continuation of this work, we suggest the following directions:

- Start from the recovery-layer primitives and find the most suitable architecture to support them.
- Implement the architecture defined in this article and perform an experimental evaluation of aspects such as overhead and response time for real-time applications.
- Select the most significant features of the architecture discussed here—rings, address spaces—and consider their explicit incorporation as features of the recovery language.
- Add these versions of the RMP to a development system for fault-tolerant software, where they could be used as building blocks to assemble user-transparent recovery mechanisms.[20]

Acknowledgments

This work was supported by a grant from the State of Florida High Technology and Industry Council and by the IBM Graduate Work Study Program.

References

1. E.B. Fernandez, R.C. Summers, and C. Wood, *Database Security and Integrity*, Addison-Wesley Publishing Co., Reading, Mass., 1981.
2. R.M. Graham, "Protection in an Information Processing Utility," *Comm. ACM*, Vol. 11, No. 5, May 1968, pp. 365-369.
3. S. Wallach and C. Holland, "32-bit Minicomputer Achieves Full 16-bit Compatibility," *Computer Design*, Vol. 20, No. 1, Jan. 1981, pp. 111-120.
4. R.E. Childs, Jr. et al., "A Processor Family for Personal Computers, *Proc. IEEE*, Vol. 72, No. 3, Mar. 1984, pp. 363-376.
5. *80286 Operating Systems Writer's Guide*, Intel Corporation, Santa Clara, Calif., 1983.
6. L. Chen and A. Avizienis, "N-version Programming: A Fault-Tolerance Approach to Reliability of Software Operation," *Digest of Eighth Intl Fault-Tolerant Computing Symp.*, IEEE, 1978, pp. 3-9.
7. B. Randell, "System Structure for Software Fault Tolerance," *IEEE Trans. Software Eng.*, Vol. SE-l, No. 3, June 1975, pp. 221-232.
8. K.H. Kim, "A Scheme for Coordinated Execution of Independently Designed Recoverable Distributed Processes," *Proc. 16th Ann. Intl Symp. of Fault-Tolerant Computing*, CS Press, Los Alamitos, Calif., 1986, pp. 130-135.
9. M.A. Ancona et al., "Using Different Language Levels for Implementing Fault Tolerant Programs," *Microprocessing and Microprogramming*, Vol. 20, 1987, pp. 33-38.
10. M.A. Ancona et al., "A System Architecture for Software Fault Tolerance," *Proc. Third Intl Conf. Fault-Tolerant Computing Systems*, Bremerhaven, W. Germany, Springer Verlag, Berlin, 1987, pp. 273-283.
11. E.B. Fernandez et al., "A System Architecture for Fault Tolerance in Concurrent Software," in preparation.
12. *iAPX 286 Architecture Extension Kernel User's Guide*, Intel Corporation, 1983.
13. K.J. Hamer Hodges, "A Fault-Tolerant Multiprocessor Design for Real-time Control," *Computer Design*, Dec. 1973, pp. 75-81.
14. iAPX 286 Hardware Reference Manual, Intel Corporation, 1983.
15. iAPX 286 Programmer's Reference Manual, Intel Corporation, 1983.
16. S.P. Morse and D.J. Albert, *The 80286 Architecture*, John Wiley and Sons, Inc., New York, 1986.
17. J.M. Westall, Jr., and A.W. Madison, "Advanced Memory Management Architectures and Virtual Machine Monitors," *Proc. 25th Ann. ACM Southeast Regional Conf.*, ACM, Apr. 1987, pp. 62-66.
18. K.H. Kim, "Approaches to Mechanization of the Conversation Scheme Based on Monitors," *IEEE Tran. Software Eng.*, Vol. SE-8, No. 3, May 1982, pp. 189-197.

19. M. McKendry and R. Campbell, "Implementing Language Support in High Level Languages," *IEEE Tran. Software Eng.*, Vol. SE-10, No. 3, May 1984, pp. 227-236.

20. J.L. Sarmiento and E.B. Fernandez, "A Knowledge-Based System for the Development of Fault-Tolerant Programs," *Proc. Florida AI Research Symp.*, M. Fishman, ed., PO Box 12560, St. Petersburg, Fla. 33733, May 1988, pp. 119-124.

Additional reading

Ozaki, B.M., *The Implementation of Software Fault Tolerance in the Intel 80286*, master's thesis, Florida Atlantic University, Boca Raton, Fla., 1987.

Brenda M. Ozaki is a senior associate engineer with IBM Corporation in the Entry Systems Division in Boca Raton, Florida. Her current interests include computer architecture, very large system integration design, and fault-tolerant systems.

Ozaki received the BSEE degree from the University of Florida and the MSCE from Florida Atlantic University.

Eduardo B. Fernandez is a professor and the associate chair of the Department of Electrical and Computer Engineering at Florida Atlantic University in Boca Raton, Florida. He has worked at the NASA Satellite Tracking Station in Santiago, Chile, taught at the University of Chile and the University of Miami, and researched the security and performance aspects of database systems at the Los Angeles Scientific Center of IBM. His current research interests are in fault-tolerant systems, computer security, database systems, and computer architecture.

Fernandez holds an MS degree in electrical engineering from Purdue University and a PhD in computer science from UCLA. He has authored three books and a large number of papers.

Ehud Gudes is a senior lecturer in the Department of Mathematics and Computer Science at Ben Gurion University in Beer-Sheva, Israel. His background includes teaching at Pennsylvania State University and managing research and development projects at Wang Laboratories, National Semiconductor, and Elron. His current research interests include database systems (emphasizing integrity), fault tolerance, data security, and expert systems.

Gudes received his BS and MS degrees from the Israel Institute of Technology and his PhD from Ohio State University.

Questions about this article can be addressed to Eduardo B. Fernandez, Department of Electrical and Computer Engineering, Florida Atlantic University, Boca Raton, FL 33431.

Performance Trade-offs for Microprocessor Cache Memories

When you consider the overhead cost of storing address tags and replacement information along with data, large block sizes make sense.

Donald B. Alpert, National Semiconductor Corporation

Michael J. Flynn, Stanford University

Improvements in very large scale integration technology have led to microprocessors with increasing logical complexity and speed. Modern microprocessors integrate cache memories with the central processing unit on chip to reduce the number of off-chip references.[1-3] A cache keeps copies of frequently used memory localities (called *blocks*) on the microprocessor chip, thus improving the cost/performance of the microprocessor and memory system.

To achieve full performance from an integrated cache memory—and the available external memory—suitable cache organization is essential. Floor planning becomes a key design element to allow for allocation of both tables and storage on a single chip. To this end, the emulation and architecture research group at Stanford studied appropriate design trade-offs and compared these trade-offs with the results of previous studies. We considered a number of questions that are central to effective cache organization:

- What is the overhead cost to store address tags and replacement information along with data in an integrated microprocessor cache memory?
- What impact does this cost have on cache performance?
- More specifically, how does altering the block size for caches occupying a constant circuit area affect performance?

Because the variation in block size directly affects the allocation of area between address tags and data in caches of constant area, we looked at issues concerning the effective combination of various types of memory in a microprocessor design.

Cache memory design for mainframes and minicomputers has been extensively studied since IBM introduced the first commercial cache in its System/360, Model 85.[4] Kenneth Kaplan and Robert Winder wrote a classic paper describing the trade-offs for mainframe caches.[5] William Strecker wrote a similar paper for minicomputer caches.[6] Here, we analyze microprocessor cache designs and compare our observations with those of previous studies.

Cache organization

A cache stores copies of main memory locations in a fast, associative buffer near the CPU. For every memory fetch, the cache is examined to

0272-1732/88/0800-0044$01.00

determine whether the reference is located in the cache. If so (a cache hit), the data is read from the cache, which avoids a slower reference to main memory. If not (a cache miss), the data is read from main memory and stored in the cache, replacing other cache data if necessary. For every memory store, the cache is also examined to determine whether the reference is located in the cache. If so, the data is written to the cache. If not, the data can either be read from memory to the cache (write allocate) or not (no write allocate). The stored data can either be written to memory immediately (write through) or written to memory later, when the data is replaced in the cache (write back).

The cache consists of several storage arrays containing a number of blocks (Figure 1). For each block, the cache stores the corresponding main memory address (the address tag) and places the data values in the corresponding main memory locations. It may store information for a replacement algorithm—or a bit or bits—indicating the validity of the address tag and the data in the block. Examples of replacement information are bits indicating the order or frequency of use of the cache blocks and a dirty bit indicating that data has been changed during its presence in a cache and must be written back to memory when the block is replaced.

When any main memory location can be stored in any cache block, the cache is fully associative. Therefore, one must include an address comparator with every block to detect a hit quickly. To reduce the number of comparators, one can restrict the placement of memory locations in the cache to a single block (direct map) or a small set of blocks (set associative).

For example, Digital Equipment Corporation's PDP 11/60 cache is a direct-map design, and both the DEC VAX 11/780 and the IBM 3033 are set-associative with two and four sets, respectively.[7,8] Direct-map and set-associative cache designs have been used in mainframes and minicomputers because the storage arrays are typically implemented with standard, static random access memory (SRAM) ICs. One can easily make integrated microprocessor caches fully associative by combining the comparator and address tag latch in a content-addressable memory cell (Figure 2). For instance, the Z80,000 microprocessor has a fully associative cache. In Figure 2, one can compare a data bit stored in the address tag latch with an inquiry data bit presented in true and complement form (Data and $\overline{\text{Data}}$). If a match is not found, the $\overline{\text{Match}}$ line is activated (Asserted Low). When Row Select is enabled (Asserted High), new data can be entered into the address tag latch from the Data and $\overline{\text{Data}}$ lines.

The most important measures of cache-design effectiveness are the miss and traffic ratios. The miss ratio is the fraction of total processor references that are missing from the cache. The traffic ratio is the number of bytes transferred between cache and main memory divided by the number of bytes transferred between the processor and memory in the absence of a cache. A low miss ratio generally indicates the cache is effective because the processor infrequently waits for slower main memory references. However, with large block sizes the traffic ratio can be high even when the miss ratio is low, indicating that the memory may be a performance bottleneck.

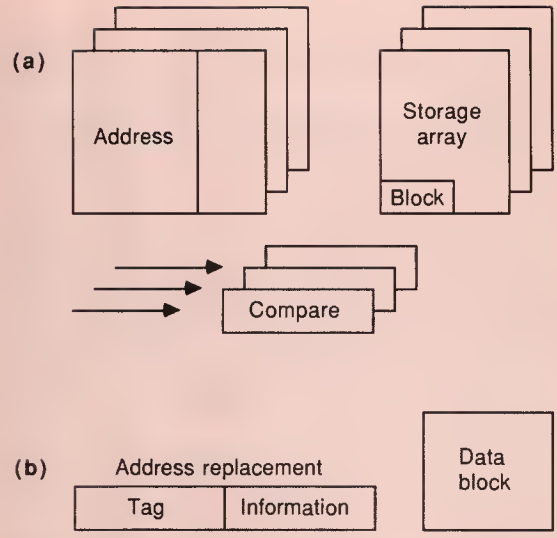


Figure 1. Set-associative cache. An address is compared to a selected entry in each address array. If a valid match is found, the word in the corresponding data storage array is selected (a). Each data block in the address array has associated storage for address and control information. This storage is the block overhead for cache data storage (b).

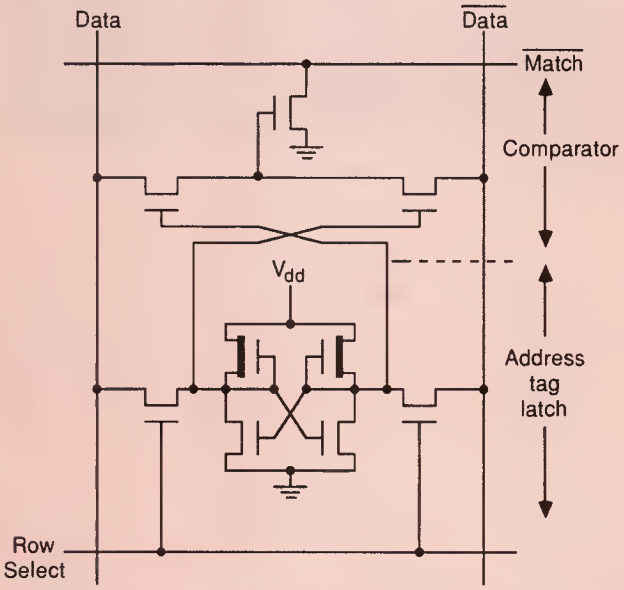


Figure 2. Content-addressed, single-bit memory circuit diagram (NMOS).

Block overhead analysis

All area measurements presented here are normalized to the area occupied by a single data bit, making it possible to examine cache organizations independently from any particular technology. If the number of data *bytes* per block is represented by b, then the cache utilization is given by the following formula:

$$\text{cache utilization} = \frac{\text{data area}}{\text{data area} + \text{block overhead}}$$

A single block of b bytes has an area of 8 b-bit area equivalents and has associated block overhead of v-bit area equivalents (see also Figure 1). These are called *area units*—or simply units—in the text.

$$\text{cache utilization} \equiv \frac{b}{\left(b + \frac{v}{8}\right)}$$

For example, if the block size is 16 bytes and the block overhead consists entirely of 32 address tag bits stored in content-addressable memory cells that occupy twice the area of a data bit (v = 64 units), then the cache utilization is two thirds.

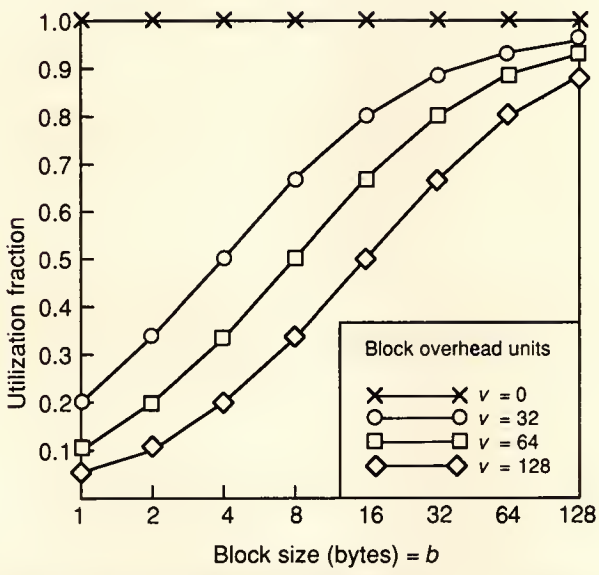


Figure 3. Cache utilization versus block size.

Alan Smith provides an excellent survey of the trade-offs of cache performance and examines cache size, block size, degree of associativity, replacement algorithm, and other design parameters.[8] Cache design studies generally vary the cache size and just one design parameter, such as block size, and then examine the effect on cache performance. Smith's studies—as well as those of Kaplan and Winder, and Strecker—compare performance results for caches with a constant quantity of data storage. In designing an integrated microprocessor cache, we believe it is better to compare performance results of caches occupying a constant circuit area. Area is the primary measure of manufacturing cost for microprocessors.

Cache utilization model

For analyzing trade-offs in cache designs occupying a constant area, it is useful to measure the cache utilization (the fraction of total cache area used to store data). The remaining area per block, called the *block overhead*, is largely occupied by the address tags, but also includes replacement information, validity bits, and buffer circuitry.

For the Z80,000 microprocessor, the cache holds 16 blocks in an area of 5.16 sq mm. Each block contains

- 16 data bytes,
- 8 data validity bits,
- 4 bits of replacement information ordering the blocks by their recency of use,
- a 28-bit address tag, and
- a tag validity bit.

The address tag and its validity bit are 28.2 percent of the total area, the replacement information is 19.1 percent, and the data validity bits are 3.1 percent. Thus, the cache utilization is the remaining area used for data, or 49.6 percent.

The block overhead, represented here by the symbol v, is measured in units of area occupied by a single data *bit*. (See accompanying box on block overhead.)

The graph in Figure 3 shows the variation of cache utilization with block size for several values of block overhead. With no block overhead (v = 0 units) the entire cache is used for data, so the utilization is always 1. The other values of block overhead roughly correspond to

- a 32-bit address tag where a tag bit and a data bit are equal in size (v = 32 units);
- a 32-bit address tag where a tag bit is twice the size of a data bit (v = 64 units); and
- a reasonable limit on the overhead, including replacement and validity bits (v = 128 units).

For the Z80,000 microprocessor cache, the block overhead is 130 units.

Table 1 shows how the number of data bytes stored in a cache varies with the block size and block overhead. The total bit area equivalent for the cache is 8,192 data bits, corresponding to 1,024 data bytes with no block overhead. For fair comparison, we rounded the number of data bytes shown in the table to the nearest whole number based on the cache utilization formula as shown. Of course, an actual cache would contain an integral number of blocks. The table shows that the amount of data in the cache varies dramatically with the block overhead for block sizes less than 16 bytes and varies less for larger block sizes. For example, with a 64-unit block overhead, the number of data bytes in the cache approximately doubles when the block size increases from 4 to 16 bytes.

To extend the concept of cache utilization just presented into a model for comparing cache designs with constant area, we made two assumptions:

- The area of a data bit remains unchanged as the block size varies.
- The block overhead remains unchanged as the block size varies.

In practice these assumptions are only roughly accurate, so the cache utilization model provides useful insight but is not the basis for a design methodology. The first assumption is inaccurate because in a practical design the area of the various storage cells is adjusted to fine-tune speed and power requirements. Unfortunately, the trade-offs between layout geometry and circuit performance cannot be analytically modeled. The second assumption is inaccurate because the number of address tag and validity bits varies with the block size. In addition, the number of replacement bits can vary with the number of blocks. In fact, the latter sources of inaccuracy are not so troublesome: The number of tag bits only changes by one for every factor-of-two difference in block size, and in some designs the area for the tags may not change at all. For example, the Z80,000 microprocessor cache stores its 28 address tag bits along a 32-bit bus connecting other functional units; the area in the four remaining bit positions is unused except for the tag validity bit. The area of the address tag is typically much greater than the sum of validity and replacement bits. Consequently, changing the number of validity and replacement bits has only a small effect on the block overhead.

Table 1. Quantity of data storage.

Block size (bytes)	Block Overhead (units) = v			
	0	32	64	128
	Cache storage available (bytes)			
2	1,024	341	205	114
4	1,024	512	341	205
8	1,024	683	512	341
16	1,024	819	683	512
32	1,024	910	819	683
64	1,024	964	910	819

Cache simulation and address traces

To compare the performance of caches with constant area and varying block size, we developed a trace of address references and used it to simulate the action of the caches. Trace-driven simulation is a standard method for evaluating memory hierarchies. Using a trace permits simulation of caches with many strategies and parameter values in reproducible experiments.

We gathered the address trace used in this study by using an instruction-set simulator for the Zilog Z8002 microprocessor.[9] We initially made traces of 15 application programs written in C that execute in user mode under Zilog's Zeus version of the Unix operating system. The traces contained records of the logical address for every instruction fetch (including immediate mode operand fetches), data fetch, and data store operation performed by the program. The traces also contained records for the system calls. Each address record indicated a 16-bit word referred to by the processor. After preliminary simulations using all test programs, we selected five typical traces for use in the study. The five programs had the following functions:

- compiler preprocessor,
- compare files,
- list a directory,
- sort a file, and
- edit a file.

We developed a composite trace by selecting 25,000 consecutive records from each of the programs in round-robin order for a total of 1,000,000 records. Each record in the composite trace was tagged to identify the program from which it was derived. Combining separate traces in this manner is an effective way to determine the typical cache performance for a work load of programs and to simulate multiprogramming. We switched the program traces every 25,000 records because this is the typical interval between context changes for a minicomputer executing an educational time-sharing work load.[10]

Constant area caches

We used the address trace described in the previous section to compare the performance of simulated caches that possessed equal areas but different block sizes. Several values of block overhead were used. The simulations assumed the caches were fully associative with a least recently used (LRU) replacement algo-

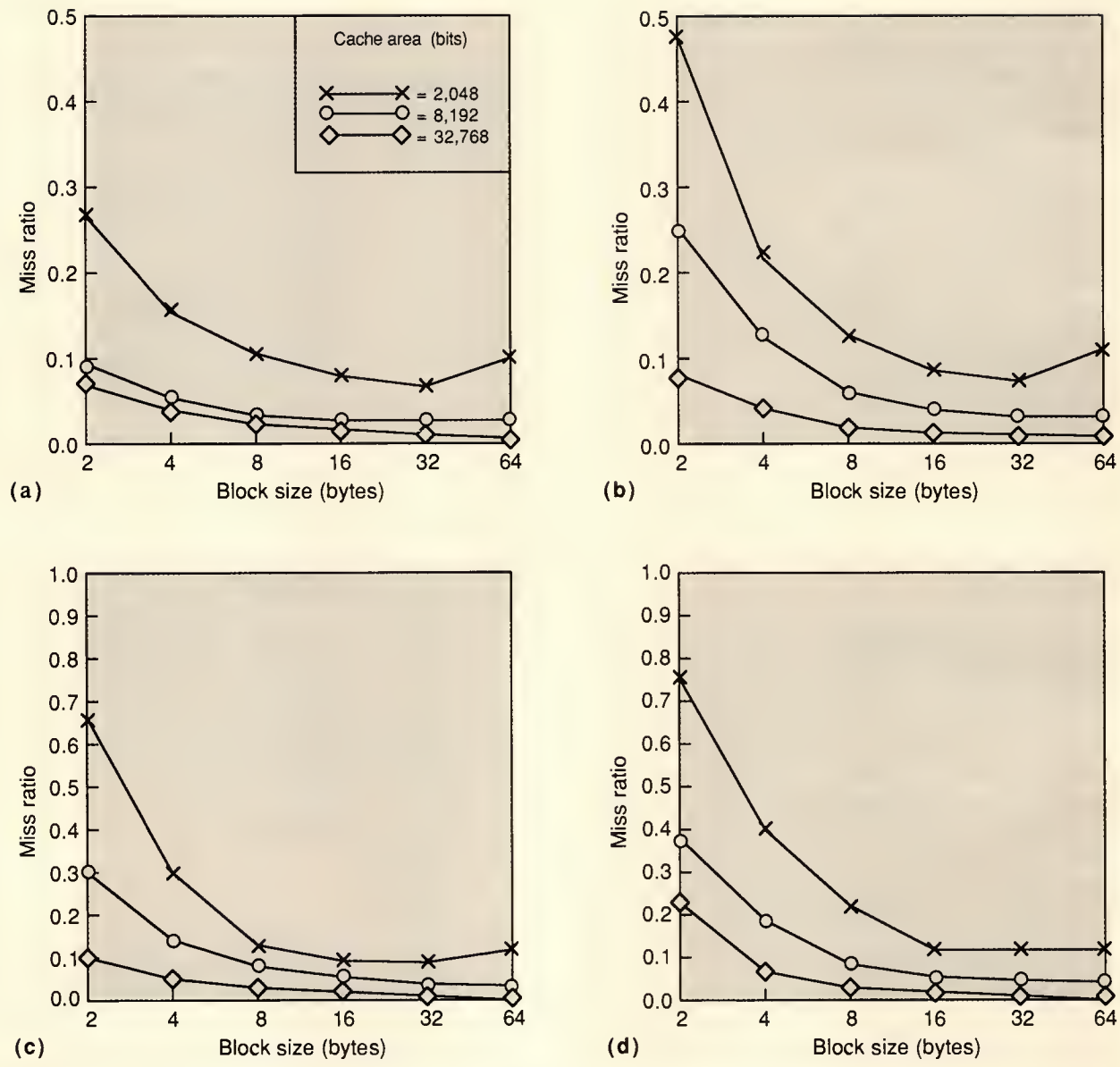


Figure 4. Miss ratios for constant area caches when block overhead is 0 units (a), 32 units (b), 64 units (c), and 128 units (d).

rithm. We selected the LRU algorithm as it generally provides the best overall cache performance. Note that the selection of a replacement algorithm generally represents a secondary effect on cache performance.[8] We used only the instruction and data fetch references in the simulation. We eliminated the data store references because cache design policies for write misses (write allocate or no write allocate) and memory update (write through or write back) are unrelated to the purpose of the study. In addition, the frequency of stores is small enough (7.5 percent) that their elimination does not affect the results of interest. (Also, for the common strategy of write through with no write allocate, the references made to memory are independent of whether writes hit or miss in the cache. Consequently, the cache's miss ratio for write references is not a very significant performance measure for such organizations.) During simulation, the cache was flushed at every system call. Although it would be preferable to flush only the fraction of the cache replaced by each particular type of system call, the information necessary to do so was unavailable. Shustek provides a description of how to gather the information when a working system is available.[11]

Figure 4 shows the variation of miss ratio with block size for simulated caches with total areas of 2,048, 8,192, and 32,768 data bits. When the block overhead is 0 units, the caches store 256, 1,024, and 4,096 bytes of data regardless of the block size. For the other values of block overhead, if the number of blocks filling the cache area is not a whole number, we used linear interpolation to estimate the miss ratio. Figure 5 shows the traffic ratio for the simulated caches.

We made three observations. First, the miss ratio becomes more sensitive to variations in block size as the block overhead increases. Thus, the graphs for small block overhead values appear relatively flatter than those for large block overhead values.

Second, for large block overhead the miss ratio decreases rapidly with increasing block size. This process results when data continues to be stored in the cache as the block size increases, particularly in the case of small block size and large block overhead. For example, when the block overhead is 0 units and the cache area is 8,192 bits, if the block size increases from 2 to 16 bytes, then the miss ratio is reduced by a factor of 3.8. But, if the block size is similarly varied for the same cache area and block overhead is 64 units, then the miss ratio is reduced by a factor of 6.5.

Third, although the miss ratio generally decreases with increasing block size, the decrease is not sufficient to prevent an increase in traffic ratio. For instance, the traffic ratio increases by 23 percent when the block overhead is 64 units and the block size is varied from 2 to 16 bytes as previously described. The increase in traffic ratio indicates that off-chip memory bandwidth can be a significant performance limitation for a microprocessor containing an integrated cache with large block size. (Notice, however, that for the smallest simulated cache sizes, the miss ratio as well as the traffic ratio may

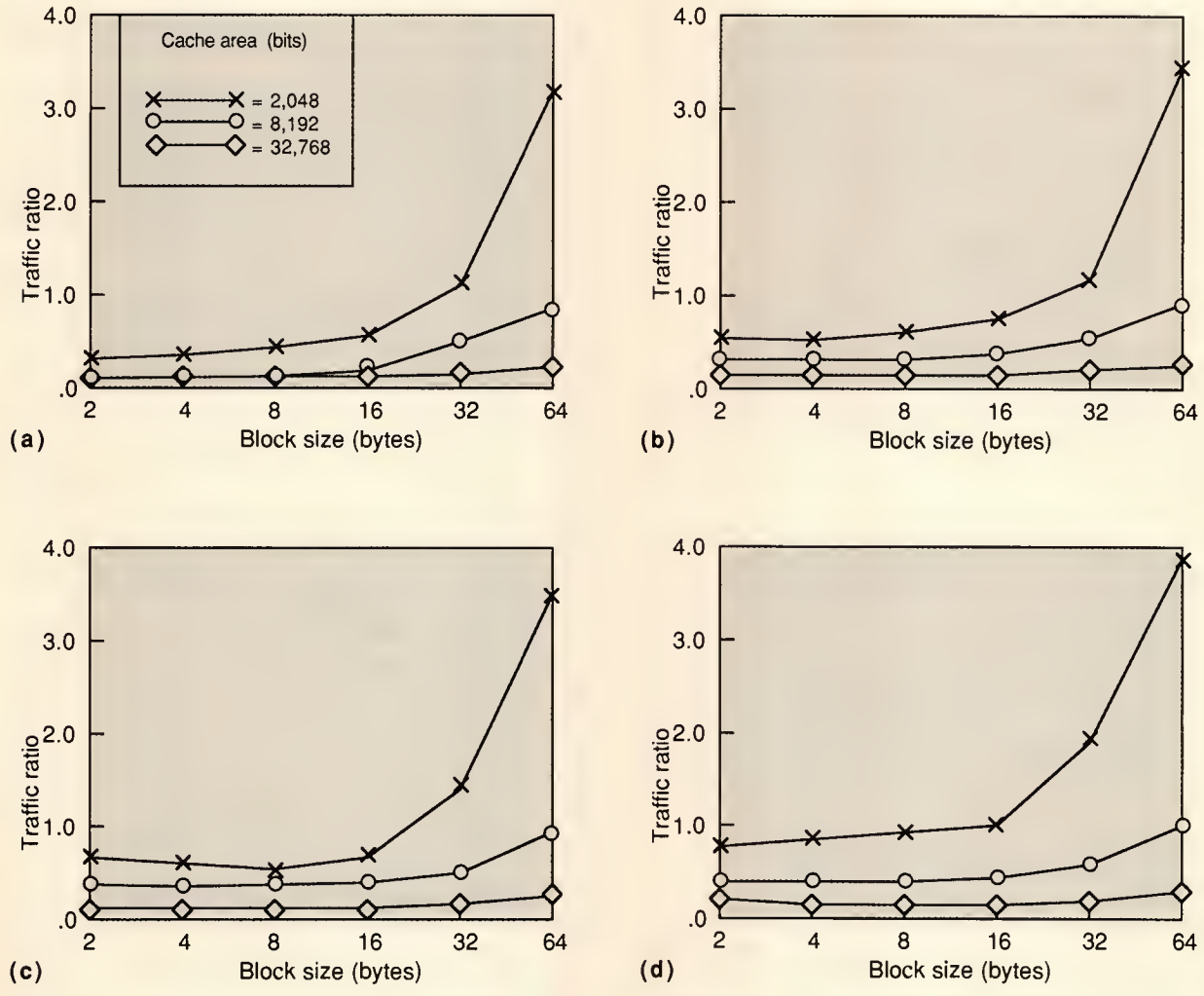


Figure 5. Traffic ratios for constant area caches when block overhead is 0 units (a), 32 units (b), 64 units (c), and 128 units (d).

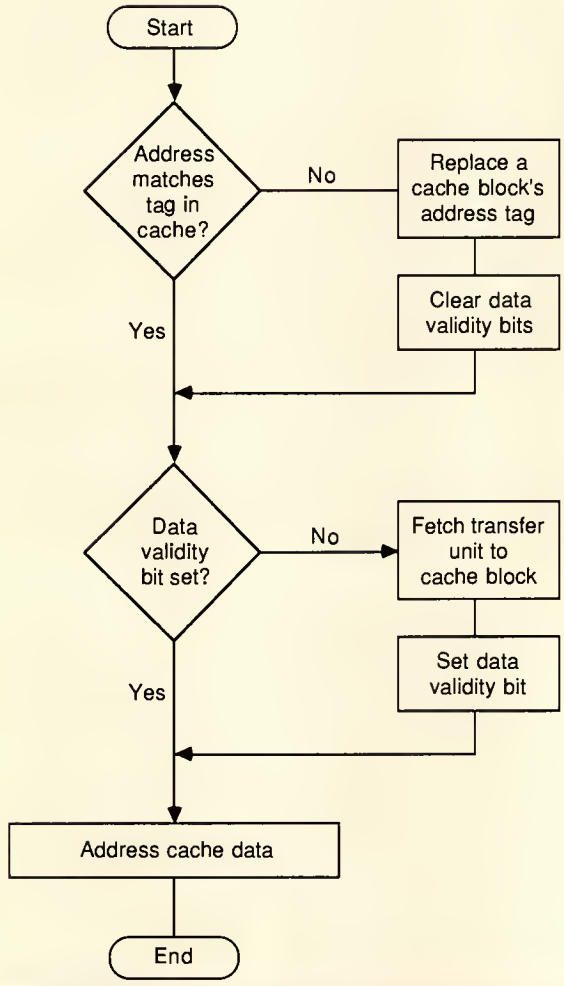


Figure 6. Flowchart for access to cache with multiple transfer units per block.

rise as the block size increases from 32 to 64 bytes. This occurs because with a block size comparable to the cache size, it becomes likely that useful information is replaced on a miss.)

These three observations suggest the wisdom of maintaining a large block size but fetching only a portion of a block when a miss occurs. This strategy may keep the miss ratio low because large blocks result in good cache utilization, yet still keep the traffic ratio low. The following section considers such a strategy.

Transfer size and block size

Most cache designs transfer an entire block when a cache miss occurs (an unsectored cache). Our simulation results indicate that fetching only a portion of the missing block may be a better strategy for integrated microprocessor caches (a sectored cache). The portion of a missing block that is fetched is called a *transfer unit*. The number of bytes in a transfer unit is called the *transfer size*; as before, the number of bytes associated with an address tag is called the block size. The strategy of fetching only a portion of a block is better for a given area and transfer size if the cache design has a lower miss ratio than a similar design with equal block and transfer sizes.

We introduce some notation to present these concepts more clearly and formally. For a cache with block overhead v, cache area A, block size b, and transfer size t, the miss ratio is represented by $m_v(A,b,t)$. The relative miss ratio, represented by α_v, is defined by the formula

$$\alpha_v \equiv \frac{m_v(A,b,t)}{m_v(A,t,t)}$$

Thus, α_v compares the miss ratios for a cache that combines several transfer units into a common block and a cache that has a single transfer unit per block. It follows from this observation: When combining several transfer units into a common block is a favorable strategy, the value of α_v is less than 1.

When the transfer size is smaller than the block size, determining whether a reference hits in the cache involves two steps, as shown in Figure 6. Initially, the address tags are examined to determine whether the addressed block is in the cache (a block hit) or not (a block miss). For a block hit, the data validity bits are examined to determine whether the addressed transfer unit is in the cache block (a transfer unit hit) or not (a transfer unit miss). When a transfer unit hit occurs on a fetch, the data referred to is read from the cache. When a transfer unit miss occurs, the missing unit is read from main memory and stored into its location in the selected block. Then the corresponding data validity bit is set. When a block miss occurs, one of the cache blocks is selected for replacement, and the address of the missing block is stored in the selected block's tag. All of the data validity bits in the block are cleared. Then the addressed transfer unit is fetched and stored in the block, and the corresponding data validity bit is set.

The IBM System/360, Model 85 uses a similar strategy.[4] The cache associates 1,024 data bytes with an address tag, and transfers 64 bytes on a miss. (In IBM design terminology a group of 64 bytes is called a *block*, and a group of 1,024 bytes is called a *sector*.) B. Rau reported that IBM selected this organization to permit fully associative tag comparison at reasonable cost even before set-associative organization was known to be effective.[12]

For cache designs containing a given quantity of data storage and using a stack replacement algorithm, the strategy of transferring partial blocks always results in a relative miss ratio of 1 or greater. (In a stack replacement algorithm, the contents of a buffer are a subset of a larger buffer's contents at all times and for all traces when the buffers are initially empty.[13] An LRU

replacement is a stack algorithm; FIFO replacement is not.) To demonstrate the relative miss ratio, consider that at some time the transfer unit addresses (t_i) that have been referred to are ordered in a stack. A cache with n blocks that each store one transfer unit contains the set of transfer units $\{t_0,t_1,\ldots,t_{n-1}\}$. If instead each block stored k transfer units (k is a factor of n), a cache that contained the same quantity of data as before would include only n/k blocks. Figure 7 shows the contents of the two caches for each of the two strategies. At any time, the second cache contains a subset of the set of transfer units in the first cache—just those transfer units included in the first $b_{n/k}$ blocks on the stack—while the contents of other transfer units are invalid. Only when the entire set of transfer units $\{t_0,t_1,\ldots,t_{n-1}\}$ is contained in $b_{n/k}$ blocks does the second cache hold all the information in the first. Consequently, for any trace, the number of misses by the second cache must be at least the number of misses by the first cache, and the relative miss ratio is at least 1.

For example, let's assume a 16-byte cache size, a 4-byte transfer size, an LRU replacement algorithm, and four references to byte addresses 0, 100, 0, and 100. With a 4-byte block size (Figure 7a with $n = 4$), a reference to byte 0 creates a miss, as does the next reference, to byte 100. The subsequent two references to bytes 0 and 100 are hits to blocks that already reside in cache. For a 16-byte block size (Figure 7b with $k = 4$ and one block, $n = k$), reference to byte 0 creates a miss that loads bytes 0 through 3 into the cache. Reference to byte 100 creates a new miss since its contents are not in the cache. Again, each subsequent reference to 0 and 100 causes a miss. An initially empty cache with a 4-byte block size has two misses, whereas a cache with a 16-byte block size has four misses. On the other hand, both caches have equal numbers of misses (two) for references to addresses 0, 12, 0, and 12.

This reasoning applies when one compares cache designs with equal quantities of data, rather than equal areas. We used the address trace previously described to compare the performance of simulated caches with equal areas and equal transfer sizes, but different block sizes. As previously shown, the simulated caches were fully associative, with LRU replacement, and only the fetch references were used. Again, we used linear interpolation to estimate the miss ratio when the number of blocks filling the cache area is not a whole number.

We performed simulations for caches with areas of 2,048, 4,096, 8,192, 16,384, and 32,768 data bits. The transfer size was varied from 2 to 16 bytes, the block size was varied from 2 to 64 bytes, and the block overhead was varied from 0 to 128 units.

Table 2 summarizes the large number of simulation results. It shows the relative miss ratio for the indicated block and transfer sizes averaged over the simulated cache areas. Thus, the reported numbers (averaged over five programs) are calculated according to the formula for mean relative miss ratio:

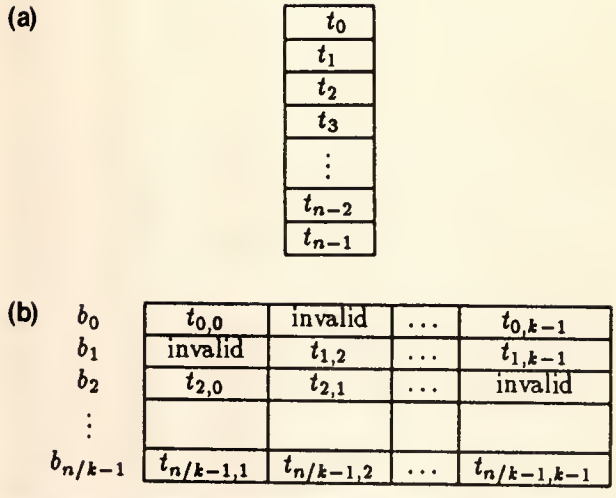


Figure 7. Comparison of cache contents for two strategies: single transfer unit per block (a), and multiple transfer units per block (b).

Table 2.
Effects of transfer size and block size on miss ratios.

Relative miss ratio = α_v						
Transfer size (bytes) = t	Block size (bytes) = b 2	4	8	16	32	64
Block overhead = 0 units						
2	1	1.043	1.101	1.199	1.478	1.744
4		1	1.055	1.154	1.430	1.713
8			1	1.102	1.379	1.682
16				1	1.250	1.563
Block overhead = 32 units						
2	1	0.876	0.821	0.840	0.903	1.062
4		1	0.936	0.959	1.040	1.247
8			1	1.026	1.124	1.372
16				1	1.101	1.388
Block overhead = 64 units						
2	1	0.789	0.680	0.661	0.692	0.816
4		1	0.847	0.824	0.865	1.038
8			1	0.973	1.032	1.268
16				1	1.069	1.348
Block overhead = 128 units						
2	1	0.746	0.635	0.538	0.550	0.632
4		1	0.856	0.733	0.747	0.864
8			1	0.849	0.869	1.022
16				1	1.031	1.246

$$\overline{\alpha_v}(b,t) \equiv \frac{1}{5}\sum_{i=1}^{5}\alpha_v(2^{10+i},b,t)$$

When the block overhead equals 0, the quantity of data storage does not vary with block size. As demonstrated, Table 2 shows the miss ratio for a given transfer size with a block overhead of 0 is consistently worse when partial blocks are fetched. For instance, if the transfer size is 4 bytes, the number of misses is 43 percent greater when the block size is 32 bytes than when the block size is 4 bytes.

But, for large values of block overhead the miss ratio can be improved by making the block size larger than the transfer size. This effect is most visible for small values of transfer size and large values of block overhead. For example, if the block overhead is 128 units and the transfer size is 4 bytes, the miss ratio decreases by 27 percent when the block size is raised from 4 to 16 bytes. The results indicate that a 2-byte block size should not be used, even when the data path to memory is only 2 bytes. When the transfer size is 2 bytes and the block overhead is as low as 32 units, the miss ratio can be improved more than 15 percent by increasing the block size from 2 to 8 or 16 bytes. For block overhead between 64 and 128 units, a block size of 16 bytes consistently provides good performance for transfer sizes between 2 and 16 bytes.

One can best see the impact of Table 2 on practical design issues by plotting this data for selected block and transfer sizes. Figures 8 and 9 show the relative miss ratio (the same as relative traffic ratio for this data) for sectored caches with constant transfer sizes of 4 and 8 bytes as a function of block overhead. The miss ratio is computed relative to an unsectored cache (Figure 7a).

For a block overhead of 0 units and a transfer of 4 bytes ($t = 4$), Figure 8 indicates the extra traffic for sectored caches. A cache with a 64-byte block size has a 73 percent higher miss ratio (and a higher traffic ratio) than the reference cache organization (4-byte block size, 4-byte transfer unit). As the block overhead increases, sectoring becomes increasingly attractive, with larger blocks ($b = 16$) overtaking ($b = 8$) by a block overhead of 64 units. Indeed, by the time the block overhead reaches 128 units, the ($b = 8$) configuration, ($b = 32$), and ($b = 16$) are almost equally attractive.

For wider transfer units ($t = 8$ bytes, see Figure 9), the same tendencies are present, but the advantages of sectoring are not as pronounced, becoming significant only at a block overhead of 128 units. While the transfer unit of 2 bytes is not plotted, Table 2 shows that it closely resembles the situation of $t = 4$ in Figure 8.

Table 1 provides the reason for the increasing advantage of sectoring as block overhead increases. As the overhead increases, fewer bits are available for data storage (the sum of cache directory and cache data storage). By sectoring the cache the directory is decreased, allowing more area for cache storage by increasing the available cache size.

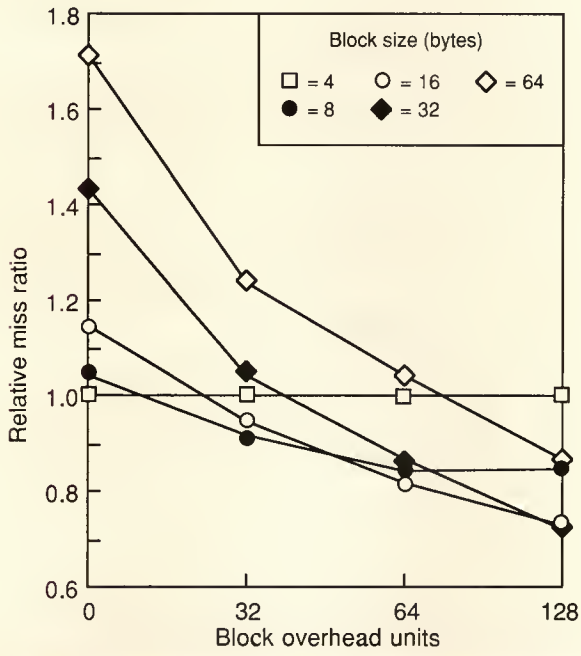


Figure 8. Miss rate relative to an unsectored cache with 4-byte block size and transfer widths.

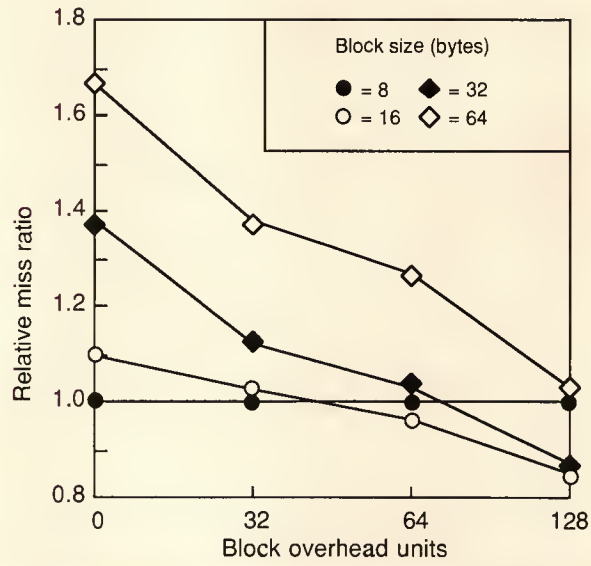


Figure 9. Miss rate relative to an unsectored cache with 8-byte block size and transfer widths.

Related research

Several other researchers have considered similar organizations for reducing the overhead of storing address tags in microprocessor cache memories. James Goodman observed both the disadvantage of transferring large blocks for small caches and the large cost of address tags for small blocks.[14] He reported favorable results for simulated caches with differing block and transfer sizes, but did not model the block overhead quantitatively. (Goodman used the terms *address block* and *transfer block* where *block* and *transfer unit* are used here.)

Hill and Smith simulated cache designs using a model similar to the cache utilization model presented here, but developed it independently.[15] They assumed that the tag and address bits are equal in area, as they are in the reduced instruction-set cache design with which they were familiar. They accounted for changes in the address tag's size with variation of the block size, but ignored replacement information and other block overhead. Based on their simulation results, Hill and Smith also recommend combining several transfer units into a single block for microprocessor designs. (Hill and Smith use the terms *block* and *subblock* in their study.)

Several microprocessors implement the strategy of fetching a partial block on a cache miss. In the Z80,000 microprocessor cache, which has a block overhead of 130 units, the transfer size is either 2 or 4 bytes and the block size is 16 bytes. In general, the Z80,000 fetches a single transfer unit for a cache miss. Only when the memory system supports a burst of several data transfers in a single bus transaction does the processor fetch an entire cache block, and then only for instruction fetch misses.

In DEC's MicroVAX chip set, eight integrated circuits implement the complete VAX architecture.[16] One circuit contains the cache's control logic and address tag array; the cache's data is stored in standard memory components. Each tag is associated with four 16-byte transfer units to reduce the area dedicated to address tags.

We have examined design trade-offs for integrated microprocessor caches. We introduced a model of cache utilization to evaluate the effects on cache performance of varying the block size. By considering the overhead cost of storing address tags and replacement information along with data, we found that large block sizes lead to more cost-effective cache designs than predicted by previous studies. When the overhead cost is high, caches that fetch only partial blocks on a miss perform better than similar caches that fetch entire blocks. More generally, this study indicates that lessons from mainframe and minicomputer design practice should be critically examined to benefit the design of microprocessors.

References

1. M. Horowitz et al., "A 32-bit Microprocessor with On-chip 2KB Instruction Cache," *ISSCC 87 Digest of Papers*, 1987, p. 30.
2. D. Archer et al., "A 32-bit CMOS Microprocessor with On-chip Instruction, Data Caching and Memory Management," *ISSCC 87 Digest of Papers*, 1987, p. 32.
3. A. Berenbaum, "A Pipelined 32-bit Microprocessor with 13KB of Cache Memory," *ISSCC 87 Digest of Papers*, 1987, p. 34.
4. J.S. Lipstay, "Structural Aspects of the System/360 Model 85, Part II: The Cache," *IBM Systems J.*, Vol. 7, No. 1, 1968, pp. 15-21.
5. K.R. Kaplan and R.O. Winder, "Cache-based Computer Systems," *Computer*, Vol. 6, No. 3, 1973, pp. 30-36.
6. W.D. Strecker, "Cache Memories for PDP-11 Family Computers," *Proc. 3rd Ann. Symp. Computer Architecture*, Jan. 1976, pp. 155-158.
7. J.C. Mudge, "Design Decisions for the PDP-11/60 Mid-range Minicomputer," *Computer Engineering*, Digital Press, Maynard, Mass., 1978.
8. A.J. Smith, "Cache Memories," *Computing Surveys*, Vol. 14, No. 3, 1982, pp. 473-530.
9. *Components Data Book*, Zilog, Inc., Santa Clara, Calif., 1983.
10. M. Satyanarayanan and D. Bhandarkar, "Design Trade-offs in VAX-11 Translation Buffer Organization," *Computer*, Vol. 14, No. 12, 1981, pp. 103-111.
11. L.J. Shustek, "Analysis and Performance of Computer Instruction Sets," SLAC Report 205, Stanford Linear Accelerator Center, Stanford University, Stanford, Calif., May 1978.
12. B.R. Rau, *Program Behavior and the Performance of Memory Systems*, doctoral dissertation, Stanford University, 1977.
13. R.L. Mattson et al., "Evaluation Techniques for Storage Hierarchies," *IBM Systems J.*, Vol. 9, No. 2, 1970, pp. 78-117.
14. J.R. Goodman, "Using Cache Memory to Reduce Processor-Memory Traffic," *Proc. 10th Ann. Symp. Computer Architecture*, June 1983, pp. 124-131.
15. M.D. Hill and A.J. Smith, "Experimental Evaluation of On-chip Microprocessor Cache Memories," *Proc. 11th Ann. Symp. Computer Architecture*, June 1984.
16. W.N. Johnson, "A Very Large Scale Integration Super-minicomputer CPU," *ISSCC Digest of Tech. Papers*, 1984, pp. 174-175.

Donald B. Alpert is architecture group manager at National Semiconductor's VLSI design center in Herzliya, Israel. His professional interests include computer architecture, microprocessor design, and memory hierarchy organization. He received the BSEE from Massachusetts Institute of Technology and the MSEE and PhD degrees from Stanford University.

Questions about this article can be directed to Michael Flynn at the Computer Systems Laboratory, Stanford University, Stanford, CA 94305-4055.

Michael J. Flynn is a professor of electrical engineering at Stanford University. His experience includes 10 years at IBM working in computer organization and design. He has also been a faculty member at Northwestern and Johns Hopkins Universities and was the director of Stanford's Computer Systems Laboratory from 1977 to 1983. His current interests include computer organization and computer architecture.

Flynn has served as vice president of the Computer Society and was a founding chair of the Technical Committee on Computer Architecture, as well as ACM's Special Interest Group on Computer Architecture.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Interest Card.

Low 159 **Medium** 160 **High** 161

GET ON THE "PAWS™ EXPRESSWAY"

TO SYSTEM DESIGN SOLUTIONS

Why not take the shortest and fastest route toward system design solutions? To define and model a large, complex system, you need a modeling tool that can address the system's total architecture.

PAWS/GPSM™ is an easy-to-use high productivity modeling and simulation tool that lets you evaluate alternative architectures while focusing on performance. It provides a state-of-the-art environment for developing accurate and reliable product definitions.

PAWS provides high level primitives closely resembling the user's intuition. PAWS models the behavior of a total system implemented in diverse technologies including software, hardware, and firmware.

GPSM, the graphical interface to PAWS, helps you quickly and easily transfer your ideas into pictures. GPSM helps you visualize multiple data and control flows through several components and incrementally synthesize models of a total system.

PAWS/GPSM is flexible. It lets you refine your model as your system design evolves so you can eliminate potential problems as early as possible in the product design cycle.

Call (512) 474-4526 to receive information on PAWS/GPSM. And get on the PAWS expressway to system design solutions.

PAWS and GPSM are trademarks of Information Research Associates, Inc.

IRA

Information Research Associates • 911 W. 29th St. • Austin, Texas 78705 • **(512) 474-4526**

Reader Service Number 1

Mathematical Software: Plod

Is user-easy software for "stiff" equations merely a fond dream? Not with the Plod approach.

Elvira Agron
National Institutes of Health

I-Lok Chang
The American University

Gamini Gunaratna

David K. Kahaner
National Bureau of Standards

Martin A. Reed
IBM Corporation

Physical models—even those described as "routine"—can pose some interesting challenges. First of all, they can involve hundreds, or thousands, of equations. These equations range from simple algebraic statements of conservation or symmetry to relations between variables at discrete time units. They include derivatives, integrals, or complicated functionals of the unknowns. Consequently, describing the model mathematically can become tedious. Another challenge is that an approximate solution, generated numerically, must eventually take a form that is palatable to users. The achievement of such a form inevitably changes the values of constants, approximations, or even the model itself.

By a *model*, we mean an initial value problem[1] given by a set of ordinary differential equations. Many models are modest, involving only a few dozen ODEs and a similar number of parameters. These models occur frequently in dynamics, such as Josephson junctions.[2] Although the problems in this group may be small, they are among the most taxing to solve numerically because of instability with respect to initial conditions, or "stiffness." (See box on the next page.) One can rarely solve models in closed form, but some of them are easy to integrate numerically. This type of model often occurs as an example in classroom situations.

Although a number of simulation software packages are available on the market, their cost, scope, capability, and quality vary tremendously. The journal *Simulation*, a good source for information on available software, has published a compendium of almost 50 such packages exclusively for microcomputers.[3] Among them, the Advanced Continuous Simulation Language package is typical and well known.[4] Some packages have good integrators, but many use explicit Runge-Kutta integration, which is inadequate for stiff ODEs.

In view of the foregoing, we designed Plod (plotted solutions of ordinary differential equations) mathematical software to address the difficulties under discussion. In addition, we wanted to provide a problem-solving tool for the expanding number of micro users unfamiliar with programming techniques. Consequently, we proceeded with the following criteria in mind:

- The physical problem can be described by 25 ODEs and 10 parameters.
- The problem is solvable while the user is at the terminal (Plod is entirely interactive).

- Plod is easy to use; no manual is required.
- The numerical methods are of high quality.
- Rapid, flexible, and attractive graphics (not publication quality) are included.
- Plod is mostly portable.
- Plod is in the public domain.

In discussion of these criteria, Plod's 25 ODE/10 parameter description addresses some application and prototype problems.[5] However, Plod is inappropriate for studying, say, the complete suspension system of an automobile. It also excludes any problems that require large amounts of CPU time. The latter conditions can occur in "chaotic" systems[2] with many abrupt changes, or when the user wants to integrate for long periods of simulated time. Plod also excludes problems that need to be repeated automatically to generate an average as a parameter is changed. We designed Plod for "what if" studies and as a prototype for more ambitious simulations.

Plod includes two files with installation and tutorial information. Users should read them, but once Plod is installed, even first-time users can proceed directly. Keyboard blunders do not prevent the package from working. It is not possible to anticipate all situations, but error recovery was a major design goal. Thus Plod is also suited for classroom use.

In regard to numerical methods, Plod's integrator, DDRIV,[6] is a modern double-precision implementation of an algorithm described by Gear.[1] This well-known algorithm is in widespread use. DDRIV implements variable-step, variable-order Adams and Gear formulas, with the ability to solve stiff ODEs. Plod's results are as accurate and reliable as current technology permits.

After reading installation and tutorial information, even first-time users can proceed directly.

To promote portability, we did not use special characteristics of a particular computer unless it was unavoidable. Plod is not entirely portable because of graphics and screen control, but we isolated these requirements and kept them to a minimum. The current target environment is an IBM PC XT/AT (discussed later), although versions have been moved to a Sun workstation and a Univac/Tektronix combination.

Instability and Stiffness

The solution curve of a specific initial value problem is insufficient in itself to determine whether the problem will be difficult to solve numerically. More important is the solution family of the ODE, that is, the set of solutions (curves) for all possible initial values. A numerical method tries to follow one member of the family (for example, the one) through the given initial point. This process can never be exact: The calculated values can be thought of as exact points on another family member. If curves in the family separate rapidly from the exact solution of the initial value problem, then no numerical method will give accurate results. Such a problem is said to be unstable with respect to initial conditions; small changes in the initial values result in a very different solution curve. Most engineering problems are, at worst, very mildly unstable in this sense.

Efficient integration of some "stiff" ODEs is a major advance. We use this term generically to describe a system that requires many small integration steps, even when the solution is slowly changing. Actually, whether a problem is stiff or not depends on a combination of factors such as the ODE, the initial conditions, the interval on which the solution is required, the numerical method employed, and seemingly minor details of implementation.

For many engineering applications, a workable notion is that a problem is stiff if it contains time constants whose values vary, at any given time, by four or more orders of magnitude. Almost every area of application leads to such problems, but one of the most common areas is chemical kinetics. Reaction rates of different species can easily vary by 10 or 20 orders. Mathematically, time constants of the model are associated with eigenvalues of the Jacobian matrix, $\delta f/\delta y$, and large differences in the magnitude of these eigenvalues are often symptomatic of stiffness. Readers can find many useful expository articles, papers, and references in *Stiff Computation*.[1]

References

1. R.C. Aiken, ed., *Stiff Computation,* Oxford, England, 1985.

Overview of Plod

Plod is a package for the solution of a set of ODEs subject to specified initial conditions,

$$\begin{aligned} y_1' &= f_1(t, y_1, y_2, \ldots, y_n) \quad & y_1(a) &= Y_1 \\ y_2' &= f_2(t, y_1, y_2, \ldots, y_n) \quad & y_2(a) &= Y_2 \\ &\vdots \\ y_n' &= f_n(t, y_1, y_2, \ldots, y_n) \quad & y_n(a) &= Y_n. \end{aligned}$$

The solution is desired on an interval $[a, b]$. Most problems also involve physical parameters whose values determine the character of the solution. The previous equation does not show these explicitly.

We designed Plod to be used in two interactive steps: Plod0 and Plod1. Plod, a batch program, runs Plod0 and optionally does the processing to pass to Plod1. During Plod0 the user enters the ODEs. No integration occurs during this step, and numerical values are not requested for any of the variables or parameters. Plod0 is a preprocessor with a Fortran program as output. The output program includes the ODEs and has a mechanism to communicate needed information to the Plod1 step.

Plod automatically compiles the output Fortran program and links it to precompiled modules supplied with the package. The result is an executable program that we call the Plod1 step. Plod1 prompts for parameter and initial values, and for the interval on which the integration is to take place. After the integration, you can generate graphs and listings. You can also change parameter values, initial conditions, and the integration interval, for example. You may experiment with the problem, examining results under different conditions. A sophisticated user can also alter the integration method and make other changes of interest to a specialist in numerical integration. During Plod1, you cannot alter the functional form of the model by adding terms, or by adding or removing equations. Those actions require returning to the Plod0 step and recompiling.

If Plod did not use a compiler, you could easily flip between changing the model and integrating the equations. The ODEs would be interpreted. We have developed a compiler-free version for our own use. Not much is lost in simple problems, but the integration of difficult problems slows by a factor of 10.

What do you need to run Plod?

Plod requires an IBM PC XT/AT (running MS-DOS) with a math coprocessor and a memory of at least 360 Kbytes. An executable file in the Plod1 step averages 270 to 300 Kbytes for five to 10 equations. Plod also runs on an IBM PC, but a hard disk is more convenient. Teachers could run Plod0, compile and link steps on an XT or AT, and distribute copies of the executable output for PC use.

Graphics necessitate an IBM enhanced graphics adapter, an IBM color graphics adapter, or a Hercules graphics card and associated monitor. Plod runs automatically at highest resolution on an EGA, CGA, or HGA, and also runs on an IBM professional graphics adapter in CGA mode. Because the HGA does not have any color capabilities and because some IBM compatibles do not support color, Plod does not utilize color in any way. An IBM graphics printer, a Proprinter, or Epson graphics printers such as models FX 80 and 100 produce screen quality plots. Printer output is optional.

A version of Plod is available to support Ryan-McFarland Fortran, version 2.14, or Lahey Fortran (F77l), version 2.20.

Linking the libraries into an executable program requires either IBM Link (version 2.3 or higher) or Phoenix Software's Plink86.

Because of the high degree of text processing, we wrote Plod0 in Pascal. However, we wrote Plod1 (mostly) in Fortran, which allowed us to make the easiest use of the high-quality ODE software currently available. Also, most scientific users already have a Fortran compiler. The need for full ANSI Fortran 77 compatibility determined our specific choices.

Please note that since we compiled the libraries supplied with Plod by using one of our supported Fortran compilers, these libraries may not link correctly with other compilers.

Portability issues. The Fortran source for Plod1 compiles with any compiler supporting the full Fortran 77 standard. The sections dealing directly with graphics are in Intel 8088 Assembler and Lattice C languages. The conventions of calling assembly language from Fortran differ for each compiler. Thus, if you request a copy of Plod you *must* specify which Fortran you are using. To generate a working copy of Plod1 from source, you must have Fortran and C compilers, as well as Assembler. To compile Plod0 requires Turbo Pascal. (To *use* Plod, you need have only a Fortran compiler.)

An example of Plod0

Plod0 has a built-in editor for generating models, which are written onto a user-specified file with extension MOD. A companion file with extension FOR for the Plod1 step is also produced. Figure 1 displays an example MOD file. MOD files use a Fortran-like syntax except that the prime symbol can denote derivatives and a percent sign can denote comments that can begin anywhere in a line. Although Figure 1 does not show it, MOD files can be quite complicated and can include IF tests. Once you have created a MOD file, you can use it as direct input to a Plod run. Thus it is only necessary for you to create the model once.

```
T
% HBRO2: Scaled concentration of Bromous Acid
% BRION: Scaled concentration of Bromide ion
% CEIV:  Scaled concentration of Cesium IV
%  T  :  Scaled seconds
% Reasonable initial conditions are
%        HBRO2=4, BRION=1.1, CEIV=4
% Reasonable parameter values are
%        S=77.270
%        Q=0.8375E-05
%        W=0.161
% A full cycle is on interval [0, 350],
%    with lots of action by T=10.
% Plot all variables, log scaling vertically
HBRO2'=S*(BRION-HBRO2*BRION+HBRO2-Q*HBRO2**2)
BRION'=(CEIV-BRION-HBRO2*BRION)/S
CEIV'=W*(HBRO2-CEIV)
```

Figure 1. Example of a MOD file generated by Plod.

An example of Plod1

A user normally selects automatic compilation and linking to Plod1 from a menu. A typical Plod1 session involves data input, integration, plot or list of results, modification of some aspect of the problem, or reexamination of the output.

You can type in initial and parameter values or read them from a file. These values include simple constants or expressions involving variables or parameters. A parser catches syntax errors or square roots of negative numbers, for example. You can also set the integration-stopping conditions. This normally involves specifying the interval on which the integration is to be performed, but it can also be stated in terms of any of the dependent variables, such as BRION = 2*HBR02.

You need not specify the method used to integrate and the accuracy of integration, but can set them in an Expert menu, which is easily accessed from the main menu. During the integration, intermediate output is displayed one screen at a time, so it is possible to verify whether things are behaving properly. The integration can also be performed "quietly." Plod takes a maximum of 400 internal steps before asking "what-next?" (The number of steps can be changed to suit specific problem needs in the Expert menu.) Even in Quiet mode, you can interrupt the calculation by pressing the space bar, and if all is well, continue the interaction. The integration step size is variable, which makes it impossible to tell in advance how many steps will be needed.

In graphics mode, up to nine curves can be drawn on one screen involving any combination of the variables, parameters, and derivatives. In practice, the most-often used plots are the dependent variables plotted against t (the default) or phase-plane plots, y_1 versus y_2. Plots can be linear or logarithmic, and can be generated on an attached printer. It is possible to move around the screen and read out the numerical values, as well as to "zoom." The first-time user can get default graphs by responding with a carriage return to every question.

Plod's integration output points drive its graphics. You need not specify the plotting interval. Choosing the output points adaptively solves an aliasing problem that is common in generating plots of dynamic variables; plots are guaranteed to be smooth and not miss structural details. To prevent too much data from being plotted, we use a sieving algorithm that only plots points that affect the appearance of the graphs. Figure 2 on the next page shows two typical plots from the model discussed.

Graphics capabilities

We examined several Fortran-callable libraries for IBM micros, but the Plod user needs to compile and link programs. Hence, the graphics library must be available to the linker. Distributing a proprietary library would have violated each of the vendors' purchase agreements. Since we were committed to development of public domain software, we saw simple graphics, or plotting, as an important need. Consequently, we wrote a small set of screen plotting subroutines.

Extensibility

Plod is a menu-driven, interactive program built on Fortran subroutines for solving ODEs. As such, it cannot be completely flexible. At the beginning of this article, we mentioned several situations where Plod is inappropriate. Sometimes the best bet is to return to the original Fortran to program a specific application. In other cases, one would like to have just a bit more flexibility than Plod allows. You can modify the Plod source, but please do not hesitate to contact us first (address at the end of this article) to see whether a more general solution could be reached that would also be useful to others.

One of the important results of the rapid spread of microprocessors is that people with less computer-oriented training are using computers to help in their jobs. This has pressured software developers into finding clever and interesting methods of generating problem-solving tools that do not require programming. Plod is one step in this direction. It is being used at several hundred sites for solving continuous time-

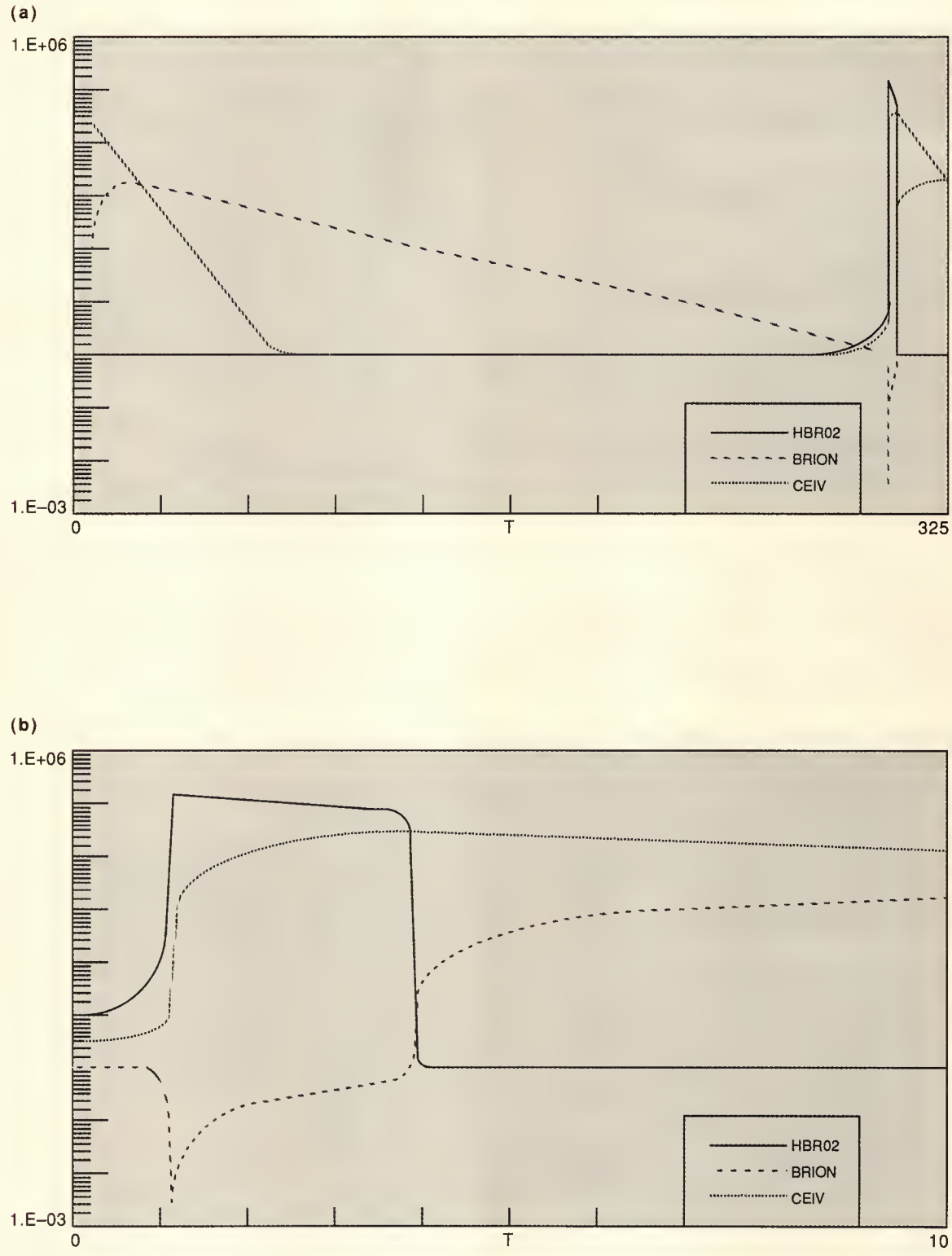


Figure 2. Field-Noyes chemical oscillator (a), and expanded view of its left edge (b).

simulation problems. As our ingenuity and experience improve, we expect to see even better products with this same nonprogramming flavor.

References

1. C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations,* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1971.
2. H. Fowler et al., "Wave Form Simulations for Josephson Junction Circuits Used for Noise Thermometry, " *NBS Tech. J.,* 1983.
3. "Catalog of Simulation Software," *Simulation,* Vol. 45, No. 4, 1985, pp. 196-209.
4. *Advanced Continuous Simulation Language (ACSL) Reference Manual,* Mitchell and Gauthier, Assoc., 290 Baker Ave., Concord, Mass., 1981, p. A-19.
5. V. Franceschini, "Bifurcations of Tori and Phase Locking in a Dissipative System of Differential Equations," *Physica,* Vol. 6D, 1983, pp. 285-304.
6. R. Boisvert, S. Howe, and D. Kahaner, "The Guide to Available Mathematical Software (GAMS)," PB 84-17135, Nat'l Tech. Info. Service, Springfield, Va., Feb. 1984.

Elvira Agron is a computer systems analyst at the National Institutes of Health, where she helps researchers utilize statistical program packages and interpret the results.

Agron received her BS degree in mathematics at the University of Puerto Rico and an MA degree in applied mathematics at the University of Maryland.

I-Lok Chang is an associate professor of mathematics at The American University, Washington, DC. His areas of interest include value distribution theory in complex variables, and optimization and approximation in numerical analysis.

Chang received his BS and PhD degrees in mathematics from the California Institute of Technology and Cornell University, respectively.

Gamini Gunaratna holds a master's degree in computer science from The American University. He has previously worked at the National Bureau of Standards and the University of Pittsburgh. His current interests include MS-DOS computers and Turbo Pascal.

David K. Kahaner is the technical group leader for scientific software and microcomputer applications in the Scientific Computing Division, Center for Applied Mathematics at the NBS. His research interests include evaluation of integrals, solution of ordinary differential equations, interpolation, Fourier transforms, and mathematical software in general.

Kahaner received his PhD in applied mathematics from the Stevens Institute of Technology, Hoboken, New Jersey. He then worked at Los Alamos National Laboratory as a numerical analyst. He has been a visiting professor in the Mathematics Department of the University of Michigan, at the ETH-Zurich, at the Technical University-Vienna, and at the Catholic University of America in Washington, DC.

Martin A. Reed is a development manager at IBM, where he has worked for the past 15 years. He received his MS degree in electrical engineering and operations research from Rensselaer Polytechnic Institute in Troy, New York. Reed holds a PhD degree in electrical engineering from the Catholic University of America, where his dissertation research involved interactive computer graphics and modeling. He is a member of the IEEE.

Questions concerning this article may be directed to David K. Kahaner, Center for Applied Mathematics, National Bureau of Standards, Technology Building, Room A161, Gaithersburg, MD 20899.

A Tutorial on CRC Computations

Data can lose integrity in storage and transmission. Here are a few software algorithms to protect your data.

Tenkasi V. Ramabadran and Sunil S. Gaitonde
Iowa State University

Cyclic redundancy codes, or CRCs, preserve the integrity of data in storage and transmission applications. CRC implementations can use either hardware or software methods. In the traditional hardware implementation, a simple shift register circuit performs the computations by handling the data a bit at a time. In software implementations, however, handling the data as bytes or even words becomes more convenient and faster. Several software algorithms for performing CRC computations have been reported in the literature.[1-3] Here, after briefly reviewing the theory behind CRC, we describe these algorithms from a mathematical viewpoint. We then compare the different algorithms in terms of their speeds and storage requirements.

Mathematical background

We shall first introduce the binary field and the binary polynomials that facilitate the definition of cyclic redundancy codes. In simple terms, a *field* is an algebraic system in which the operations of addition, subtraction, multiplication, and division can be performed. The set of real numbers, for example, forms a field. Fields can be *finite* or *infinite*. The smallest finite field is the binary field that has just two elements denoted usually by 0 and 1. The tables in Figure 1 define the addition and multiplication operations in this field.

From the addition table, we see that an EXOR gate is all that we need to perform the addition operation in the binary field. Moreover, we see 0 and 1 to be their own additive inverses, and so subtraction in the binary field is the same as addition. Multiplication in the binary field can be performed simply by means of an AND gate. We must define division in this field only for the single nonzero element 1, and we do this trivially by noting that division by 1 leaves both 0 and 1 unchanged.

A *binary polynomial* is a polynomial with coefficients from the binary field. For example, $0, 1, x, 1 + x, x^2, 1 + x + x^2$ are all binary polynomials in the dummy variable x. Given any sequence of bits, we can associate a binary polynomial with it by regarding the different bits as representing the coefficients of the polynomial. For instance, with the sequence 1 0 1 0 1 1, we can associate the fifth degree polynomial $1 \cdot x^0 + 0 \cdot x^1 + 1 \cdot x^2 + 0 \cdot x^3 + 1 \cdot x^4 + 1 \cdot x^5 = 1 + x^2 + x^4 + x^5$.

0272-1732/88/0800-0062$01.00 © 1988 IEEE

According to the convention used here, the rightmost bit of a sequence represents the coefficient of the highest degree term of the associated polynomial. A left-to-right shift of a sequence of bits by i positions (with the vacated positions filled with 0's), therefore, corresponds to multiplying the associated polynomial by x^i. We perform operations involving binary polynomials in exactly the same manner as we do with ordinary polynomials, that is, polynomials with real number coefficients. However, we manipulate the coefficients using the rules of the binary field. Figure 2 contains some examples.

Cyclic redundancy codes

Error control coding provides the means to protect data from errors and involves essentially adding a certain amount of redundancy to the data in a controlled fashion. In a typical (block) coding scheme, the data to be protected is first divided into k-bit *message* blocks. Each block is then encoded into an n-bit ($n > k$) *codeword*. The redundancy added therefore amounts to $n - k$ bits per message block. We refer to these bits as *check* (or *parity*) bits.

We collectively refer to the set of codewords as a *code* and select it such that it has good error correction/detection capabilities and also some algebraic structure to facilitate implementation. A simple example of a code is the even-parity code used in many microcomputer systems to protect data in memory. For this code, $k = 8$, $n = 9$, and $n - k = 1$. All codewords have even parity, and the code is capable of detecting all odd numbers (1, 3, 5, 7, and 9) of errors in a codeword.

+	0	1
0	0	1
1	1	0

(a)

·	0	1
0	0	0
1	0	1

(b)

Figure 1. Addition table (a) and multiplication table (b).

The selection of a code for a specific application depends on a number of factors including the amount of protection required, the overhead involved, the cost of implementation, the error control strategy employed, and the nature of errors. The $n - k$ check bits of a code forming the overhead directly affect the error control capability of the code and thereby the amount of protection provided. In general, the more check bits in a code, the greater is its power of error correction/detection. For a given number of check bits, the relative overhead of a code can be kept low by using a large message block size k; however, this action tends to increase the implementation cost of the code.

Two error control strategies have been popular in practice. They are the FEC, or forward error correction, strategy, which uses error correction alone, and the ARQ, or automatic repeat request, strategy, which uses error detection combined with retransmission of corrupted data. The ARQ strategy is generally preferred for several reasons, including the fact that the number of check bits required to provide a certain amount of error protection is smaller for error detection than

$$1 + x + x^3 + x^6$$
$$x + x^2 + x^4 + x^6$$
$$1 + x^2 + x^3 + x^4$$

Addition (or subtraction)

$$(1 + x^2 + x^4) \cdot (x + x^3) = x + x^7$$

Multiplication

$$\begin{array}{r|l} & x^2 + x + 1 \quad \text{←Quotient} \\ x^4 + x^2 + 1 & x^6 + x^5 + x + 1 \\ & x^6 + x^4 + x^2 \\ \hline & x^5 + x^4 + x^2 + x + 1 \\ & x^5 + x^3 + x \\ \hline & x^4 + x^3 + x^2 + 1 \\ & x^4 + x^2 + 1 \\ \hline & x^3 \quad \text{←Remainder} \end{array}$$

Division

Figure 2. Examples of operations involving binary polynomials.

for error correction. The FEC strategy is typically used when retransmission is impossible or impractical. Depending on the nature of the error-generating mechanism in the storage or transmission medium, errors can be *random* (isolated) or *bursty* (clustered) and correspondingly influence the choice of a code.

Cyclic redundancy codes (also known as *polynomial* codes) form a powerful class of codes suited especially for the detection of burst errors. In such a code, we select the codewords such that the associated polynomials are multiples of a certain polynomial $g(x)$ called the *generator polynomial*. The generator polynomial therefore decides the error control properties of a CRC. Later, we discuss how we can choose the generator polynomial of a CRC in such a way that we induce certain desirable properties in the code. Some commonly used generator polynomials are listed in Table 1.

We now present a simple encoding procedure for a CRC given its generator polynomial $g(x)$ of degree $n-k$. Let $u(x)$ and $v(x)$ represent the polynomials associated with a k-bit message and the corresponding n-bit codeword respectively. We refer to $u(x)$ and $v(x)$, with respective (maximum) degrees of $k-1$ and $n-1$, as a *message polynomial* and a *codeword polynomial* respectively. The relationship between $u(x)$ and $v(x)$ is then expressed as

$$v(x) = u(x)\ g(x).$$

Notice that the number of added redundant bits equals the degree of the generator polynomial $g(x)$. The above encoding procedure, which involves the multiplication of two binary polynomials, can easily be implemented in hardware by means of shift registers and EXOR gates.

We describe another encoding procedure for a CRC by the equation

$$v(x) = s(x) + x^{n-k}\,u(x)$$

where $s(x) = R_{g(x)}[x^{n-k}\,u(x)]$ is the remainder resulting from the division of $x^{n-k}u(x)$ by $g(x)$. That is, for some unique polynomials $a(x)$ and $s(x)$, we can write

$$x^{n-k}u(x) = a(x)g(x) + s(x)$$

so that $v(x) = a(x)g(x)$ is a multiple of $g(x)$.

In this encoding procedure the rightmost k bits of a codeword, which correspond to the coefficients of $x^{n-k}\,u(x)$, are simply the message bits being encoded. The leftmost $n-k$ bits, which correspond to the coefficients of $s(x)$ (which has a maximum degree of $n-k-1$), form the check bits. Later, we discuss the implementation of this encoding procedure using hardware and software approaches.

Both of the encoding procedures we described generate the same set of codewords [multiples of $g(x)$], but the correspondence between a message and a codeword is altered depending on the procedure used. The second procedure, which maps a message into a codeword containing the message itself as a constituent part, is invariably used because the encoding and decoding algorithms are essentially identical when using this procedure.

Let us look at an example of this procedure. Let the message to be encoded be the sequence 1 0 1 0 1 1 represented by the polynomial $u(x) = 1 + x^2 + x^4 + x^5$. Let $g(x)$ be the CRC-16 polynomial $1 + x^2 + x^{15} + x^{16}$. Then

$$x^{n-k}u(x) = x^{16}(x^5 + x^4 + x^2 + 1)$$

$$= x^{21} + x^{20} + x^{18} + x^{16}.$$

Dividing $x^{n-k}u(x)$ by $g(x)$, we obtain the remainder $s(x)$ seen in Figure 3.

The check bits are therefore 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0, and the message 1 0 1 0 1 1 is encoded into the codeword 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 1 1.

We accomplish error detection using a CRC as follows. Let a message polynomial $u(x)$ be encoded into

Table 1.
Commonly used generator polynomials.

CRC-16	$x^{16} + x^{15} + x^2 + 1$
SDLC (IBM, CCITT)	$x^{16} + x^{12} + x^5 + 1$
CRC-16 REVERSE	$x^{16} + x^{14} + x + 1$
SDLC REVERSE	$x^{16} + x^{11} + x^4 + 1$
LRCC-16	$x^{16} + 1$
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + x + 1$
LRCC-8	$x^8 + 1$
ETHERNET	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

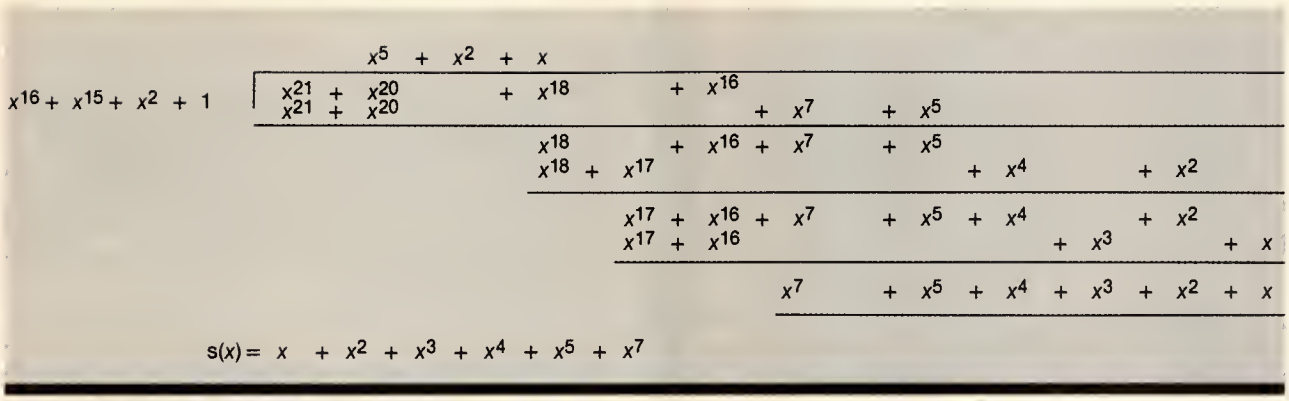


Figure 3. An example of message encoding.

the codeword polynomial $v(x)$ before transmission or storage. Let the errors affecting the codeword be represented by the *error polynomial* $e(x)$. The maximum degree of $e(x)$ is $n-1$, and its nonzero terms indicate the positions where errors have occurred.

The polynomial $r(x) = v(x) + e(x)$ represents the *corrupted* codeword received at the decoder and has a maximum degree also of $n-1$. The decoder checks whether $r(x)$ represents a codeword, that is, whether it is a multiple of $g(x)$. We do this simply by dividing $r(x)$ (or equivalently $[x^{n-k} r(x)]$ by $g(x)$ and checking for a zero remainder. If the remainder is zero, we assume $r(x)$ to be error free. That is, $e(x) = 0$ so that $r(x) = v(x)$, and the original message $u(x)$ may be recovered from it.

If the remainder is not zero, we detect the presence of one or more errors. However, the possibility exists that the remainder is zero; that is, $r(x)$ is a multiple of $g(x)$, even though errors have occurred. When can this happen? Since both $v(x)$ and $r(x)$ are multiples of $g(x)$, such a situation arises when $e(x)$ itself is a nonzero multiple of $g(x)$; that is, when $e(x)$ represents a nonzero codeword. The probability of such *undetected* errors depends on the nature and distribution of errors in the storage or transmission medium and the generator polynomial $g(x)$ of the CRC.

We can control the error detection capability of a CRC by suitable choice of its generator polynomial $g(x)$.[4] Consider a *single* error pattern represented by $e(x) = x^i$ for some i, $0 \le i \le n-1$. If $g(x)$ has more than one nonzero term, it does not divide x^i evenly and therefore can detect all single errors.

Suppose $g(x)$ has $(1 + x)$ as one of its factors. Then the codeword polynomials, which are multiples of $g(x)$, also have $(1 + x)$ as a factor. Any polynomial with $(1 + x)$ as a factor has an even number of terms. We show this by considering some polynomial $v(x) = (1+x)\, w(x)$. Upon substituting $x = 1$, we have $v(1) = (1 + 1)w(1) = 0$, implying that $v(x)$ has an even number of terms. Therefore, if $g(x)$ has $(1 + x)$ as a factor, all the codewords have even parity and any *odd* number of errors can be detected.

Consider now a *double* error pattern $e(x) = x^i + x^j = x^i (1 + x^{j-i})$ for some i, $0 \le i \le n-2$ and j, $i+1 \le j \le n-1$. If $g(x)$ does not have x as a factor and if it does not evenly divide $[1 + x^{j-i}]$ for $1 \le j - i \le n-1$, we can detect all double errors.

Let us now consider the detection of burst errors. A *burst* error of length b is any error pattern for which the number of bits between the first and last errors, including these errors, is b. For example, the error pattern 0 0 0 1 0 1 0 1 1 0 0, ..., 0 represented by $e(x) = x^3 + x^5 + x^7 + x^8$ is a burst error of length 6. Let the generator polynomial of a CRC be of the form $g(x) = 1 + g_1 x + \cdots + g_{n-k-1} x^{n-k-1} + x^{n-k}$ where $g_1, g_2, \ldots, g_{n-k-1}$ can be either 0 or 1. In other words, $g(x)$ has a degree of $n-k$ and is not divisible by x. Any burst error of length $(n-k)$ or less can be represented as $e(x) = x^i (1 + e_1 x + \cdots + e_{n-k-1} x^{n-k-1})$ for some i, $0 \le i \le k$ and where $e_1, e_2, \ldots, e_{n-k-1}$ can be either 0 or 1. Clearly, such a polynomial is not evenly divisible by $g(x)$, and therefore we can detect the corresponding burst error.

Consider now a burst error of length $n-k+1$ represented by $e(x) = x^i (1 + e_1 x + \cdots + e_{n-k-1} x^{n-k-1} + x^{n-k})$. Of the 2^{n-k-1} possible error patterns of this form for each i, $0 \le i \le k-1$, only one error pattern, namely, $e(x) = x^i g(x)$, is undetectable. The fraction of undetected burst errors of length $n-k+1$ is therefore $2^{-(n-k-1)}$. Similar consideration shows that the fraction of undetected burst errors of length greater than $n-k+1$ is $2^{-(n-k)}$. Notice the fundamental role played by the number of check bits $n-k$ in the detection of burst errors.

We now quantify the error detection capability of a specific code, the one generated by the CRC-16 polynomial. For this code, $g(x) = 1 + x^2 + x^{15} + x^{16} = (1+x)\,(1+x+x^{15})$ and the smallest integer m for which $g(x)$ divides $[1 + x^m]$ is 32,767. So, if the codeword length $n \le 32{,}767$ (or equivalently, if the message block size $k \le 32{,}751$), the CRC-16 polynomial can detect all single, double, triple, and odd numbers of errors. Furthermore, it can detect all burst errors of length 16 or less, 99.997 percent of 17-bit error bursts, and 99.998 percent of 18-bit or longer error bursts.

Among the different generator polynomials shown earlier in Table 1, the CRC-16 polynomial is commonly used worldwide; for instance, the Bisync (binary synchronous) protocol uses it. IBM's synchronous data

link control protocol uses the SDLC polynomial, and CCITT (the International Consultative Committee for Telegrapy and Telephony) has standardized it. The "reverse" polynomials are the same as the "forward" polynomials except that the codewords are in reverse order. The CRC-12 polynomial is used with 6-bit characters. The LRC (Longitudinal Redundancy Code) polynomials are used for simple longitudinal parity calculations, that is, a mod-2 sum of bytes (LRCC-8) or words (LRCC-16). The Ethernet polynomial is used in local area networks.[5]

Hardware implementation

The second CRC encoding procedure described earlier involves the computation of $s(x)$, which is the remainder resulting from the division of $x^{n-k}u(x)$ by $g(x)$. The decoding operation also involves the division of $r(x)$ [or $x^{n-k}r(x)$] by $g(x)$ for computing the remainder. A hardware circuit that simulates the division operation is a linear feedback shift register.[4] Figure 4 shows the LFSR circuit for the CRC-16 polynomial. Notice that the message polynomial $u(x)$ [$r(x)$ for decoding] feeds in at the right end of the shift register that corresponds to multiplication of $u(x)$ [or $r(x)$] by x^{n-k}.

We perform CRC encoding using a LFSR circuit as follows. We feed in the entire message $u(x)$ (with the rightmost bit first) at the right end of the LFSR circuit. The circuit performs the division operation and stores the check bits corresponding to $s(x)$ in the different shift register stages as indicated. Then we append these bits to the message to form the codeword. The decoding operation is quite similar. We feed the received polynomial $r(x)$ into the LFSR circuit to compute the remainder. If the remainder is zero, we simply drop the check bits from the codeword to recover the message. If the remainder is not zero, we have detected the presence of error(s) and must take suitable action to recover from the error condition.

Software implementations

In the hardware implementation we just described, we process data bit by bit. It is possible to duplicate the operation of the hardware circuit in software, giving rise to what we refer to as the bitwise, or CRCB, algorithm. However, we can achieve faster implementations in software by handling the data as bytes or even words. We derive the necessary relationship for such implementations by considering the effect on the check bits of a message when the message is augmented by a byte. We assume a generator polynomial $g(x)$ of degree 16 (for example, CRC-16) in the following derivation.

Let $u(x)$ and $s(x)$ represent a message and the corresponding check bits respectively. These binary polynomials are of the form,

$$u(x) = u_0 + u_1x + u_2x^2 + \cdots, \text{ and}$$

$$s(x) = s_0 + s_1x + s_2x^2 + \cdots + s_{15}x^{15}.$$

Since $s(x)$ is the remainder resulting from the division of $x^{16}u(x)$ by $g(x)$, we can write for some polynomial $a(x)$,

$$x^{16}u(x) = a(x)g(x) + s(x).$$

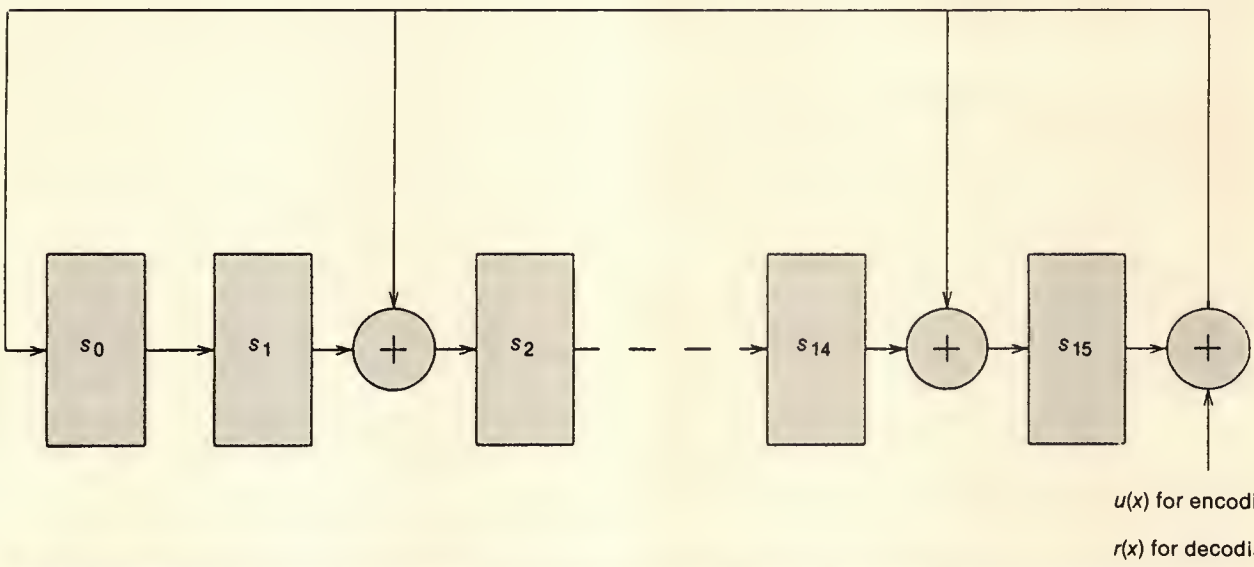


Figure 4. LFSR circuit for the CRC-16 polynomial $1 + x^2 + x^{15} + x^{16}$.

Let us now augment the message by one more byte. This corresponds to shifting the original message eight positions to the right and inserting the extra byte in the vacated positions. The augmented message $u'(x)$ can therefore be written as

$$u'(x) = b(x) + x^8 u(x) \text{ where,}$$
$$b(x) = b_0 + b_1 x + \cdots + b_7 x^7$$

corresponds to the newly added byte. Let $s'(x)$ represent the check bits corresponding to $u'(x)$. Then, $s'(x)$ is the remainder resulting from the division of $x^{16}u'(x)$ by $g(x)$. We will denote this symbolically as

$$s'(x) = R_{g(x)}[x^{16}u'(x)].$$

Expressing $u'(x)$ in terms of $u(x)$, we have

$$\begin{aligned} x^{16}u'(x) &= x^{16}[b(x) + x^8 u(x)] \\ &= x^{16}b(x) + x^8[x^{16}u(x)] \\ &= x^{16}b(x) + x^8[a(x)g(x) + s(x)] \\ &= x^{16}b(x) + x^8 a(x)g(x) + x^8 s(x) \end{aligned}$$

The remainder of a sum of polynomials under division by another polynomial is simply the sum of the remainders of the individual polynomials. So,

$$\begin{aligned} s'(x) &= R_{g(x)}[x^{16}u'(x)] \\ &= R_{g(x)}[x^{16}b(x) + x^8 a(x)g(x) + x^8 s(x)] \\ &= R_{g(x)}[x^{16}b(x)] + R_{g(x)}[x^8 a(x)g(x)] \\ &\quad + R_{g(x)}[x^8 s(x)] \\ &= R_{g(x)}[x^{16}b(x) + x^8 s(x)] \end{aligned}$$

since $g(x)$ evenly divides $x^8 a(x)g(x)$. Expressing $b(x)$ and $s(x)$ in expanded form,

$$\begin{aligned} &x^{16}b(x) + x^8 s(x) = b_0 x^{16} + b_1 x^{17} + \cdots + b_7 x^{23} \\ &+ s_0 x^8 + s_1 x^9 + \cdots + s_7 x^{15} \\ &+ s_8 x^{16} + s_9 x^{17} + \cdots + s_{15} x^{23} \\ &= (b_0 + s_8)x^{16} + (b_1 + s_9)x^{17} + \cdots + (b_7 + s_{15})x^{23} \\ &+ s_0 x^8 + s_1 x^9 + \cdots + s_7 x^{15}. \end{aligned}$$

Replacing $[b_i + s_{i+8}]$ by t_i for $i = 0, 1, \ldots, 7$,

$$\begin{aligned} &x^{16}b(x) + x^8 s(x) = t_0 x^{16} + t_1 x^{17} + \cdots + t_7 x^{23} \\ &+ s_0 x^8 + s_1 x^9 + \cdots + s_7 x^{15} \end{aligned}$$

and

$$\begin{aligned} s'(x) &= R_{g(x)}[t_0 x^{16} + t_1 x^{17} + \cdots + t_7 x^{23}] \\ &+ R_{g(x)}[s_0 x^8 + s_1 x^9 + \cdots + s_7 x^{15}] \\ &= R_{g(x)}[t_0 x^{16} + t_1 x^{17} + \cdots + t_7 x^{23}] \\ &+ (s_0 x^8 + s_1 x^9 + \cdots + s_7 x^{15}) \end{aligned}$$

since the degree of the second expression is smaller than that of $g(x)$.

The last equation relates the check bits of the augmented message with the check bits of the original message and the added byte. Notice that the expression $(s_0 x^8 + s_1 x^9 + \cdots + s_7 x^{15})$ represents the high-order eight check bits of $u(x)$ shifted to the right by eight positions. The bits $(t_0, t_1, \ldots, t_7)$ are simply the sum of the added byte $(b_0, b_1, \ldots, b_7)$ and the low-order eight check bits $(s_8, s_9, \ldots, s_{15})$ of $u(x)$.

Different algorithms for CRC computations may be viewed as methods to compute $s'(x)$ from $s(x)$ and $b(x)$ using the last equation. In the following material, we present a few bytewise algorithms that can be used directly with generator polynomials of degree 16. Modifications to these algorithms to handle generator polynomials of other degrees and to handle other units of data, for example, words, are fairly straightforward.

Table lookup algorithm

For different values of the byte $(t_0, t_1, \ldots, t_7)$, we can precompute the values of the remainder $R_{g(x)}[t_0 x^{16} + t_1 x^{17} + \cdots + t_7 x^{23}]$ and store them in a table. Such a table would have 256 entries, each two bytes long. For example, Table 2 stores the lookup values corresponding to the CRC-16 polynomial. We describe an algorithm for CRC computations using such a table as follows. Assume that the check bits are stored in a register referred to as the CRC register.

1) Initialize the CRC register to 0000 hex, that is, set the values of s_0 through s_{15} to 0.
2) EXOR the input byte $(b_0, b_1, \ldots, b_7)$ with $(s_8, s_9, \ldots, s_{15})$ to form $(t_0, t, \ldots, t_7)$.
3) Shift the CRC register eight positions to the right.
4) Look up the value corresponding to $(t_0, t_1, \cdots, t_7)$ from the table and EXOR the CRC register with it.
5) Repeat steps 2 to 4 until you reach the end of the message.

Compared with the bitwise algorithm, the table lookup, or CRCT, algorithm we described has a considerable speed advantage.

Reduced table lookup algorithm

In some applications the amount of storage required for the table lookup algorithm may be too high. We can use a reduced (smaller) table in such situations. The basic idea here is to split up the expression $R_{g(x)}[t_0 x^{16} + t_1 x^{17} + \cdots + t_7 x^{23}]$ into the sum $R_{g(x)}[t_0 x^{16}] + R_{g(x)}[t_1 x^{17}] + \cdots + R_{g(x)}[t_7 x^{23}]$. For $i = 0, 1, \ldots, 7$, t_i can be either 0 or 1. If $t_i = 0$, $R_{g(x)}[t_i x^{i+16}] = 0$. So, we must precompute only eight 16-bit values corresponding to $R_{g(x)}[x^{16}]$, $R_{g(x)}[x^{17}]$, $\cdots$, and $R_{g(x)}[x^{23}]$. These values can be stored in a smaller table just 16 bytes long as against the 512 bytes required for the previous algorithm. Table 3 shows such a reduced table with precomputed values for CRC-16. We describe the bytewise reduced table lookup, or CRCR(B), algorithm as follows:

Table 2.
Lookup table for CRC-16.

0000	D801	F001	2800	A001	7800	5000	8801
C0C1	18C0	30C0	E8C1	60C0	B8C1	90C1	48C0
C181	1980	3180	E981	6180	B981	9181	4980
0140	D941	F141	2940	A141	7940	5140	8941
C301	1B00	3300	EB01	6300	BB01	9301	4B00
03C0	DBC1	F3C1	2BC0	A3C1	7BC0	53C0	8BC1
0280	DA81	F281	2A80	A281	7A80	5280	8A81
C241	1A40	3240	EA41	6240	BA41	9241	4A40
C601	1E00	3600	EE01	6600	BE01	9601	4E00
06C0	DEC1	F6C1	2EC0	A6C1	7EC0	56C0	8EC1
0780	DF81	F781	2F80	A781	7F80	5780	8F81
C741	1F40	3740	EF41	6740	BF41	9741	4F40
0500	DD01	F501	2D00	A501	7D00	5500	8D01
C5C1	1DC0	35C0	EDC1	65C0	BDC1	95C1	4DC0
C481	1C80	3480	EC81	6480	BC81	9481	4C80
0440	DC41	F441	2C40	A441	7C40	5440	8C41
CC01	1400	3C00	E401	6C00	B401	9C01	4400
0CC0	D4C1	FCC1	24C0	ACC1	74C0	5CC0	84C1
0D80	D581	FD81	2580	AD81	7580	5D80	8581
CD41	1540	3D40	E541	6D40	B541	9D41	4540
0F00	D701	FF01	2700	AF01	7700	5F00	8701
CFC1	17C0	3FC0	E7C1	6FC0	B7C1	9FC1	47C0
CE81	1680	3E80	E681	6E80	B681	9E81	4680
0E40	D641	FE41	2640	AE41	7640	5E40	8641
0A00	D201	FA01	2200	AA01	7200	5A00	8201
CAC1	12C0	3AC0	E2C1	6AC0	B2C1	9AC1	42C0
CB81	1380	3B80	E381	6B80	B381	9B81	4380
0B40	D341	FB41	2340	AB41	7340	5B40	8341
C901	1100	3900	E101	6900	B101	9901	4100
09C0	D1C1	F9C1	21C0	A9C1	71C0	59C0	81C1
0880	D081	F881	2080	A881	7080	5880	8081
C841	1040	3840	E041	6840	B041	9841	4040

Note: The lookup values in hex notation correspond to $(t_0, t_1, \ldots, t_7)$, ranging from 0 to 255 and increasing first by column and then by row.

1) Initialize the CRC register to 0000 hex.

2) EXOR the input byte $(b_0, b_1, \ldots, b_7)$ with $(s_8, s_9, \ldots, s_{15})$ to form $(t_0, t_1, \ldots, t_7)$.

3) Shift the CRC register eight positions to the right.

4) For $i = 0, 1, \ldots, 7$ if $t_i = 1$, look up the corresponding 16-bit value $(R_{g(x)}[x^{i+16}])$ from the table and EXOR the CRC register with it.

5) Repeat steps 2 to 4 until you reach the end of the message.

Notice that only step 4 differs from the previous algorithm. Instead of a single EXOR operation, the number of EXOR operations corresponds to the number of nonzero bits in $(t_0, t_1, \ldots, t_7)$. An average value of 4 seems reasonable for random data. This increased number of operations naturally detracts from the speed performance of the algorithm.

It is possible to generate a regular lookup table from a reduced lookup table in a simple manner. For example, Table 2 can be generated from Table 3 as follows. For any given byte $(t_0, t_1, \ldots, t_7)$, generate the lookup value in Table 2 by EXORing the entries of Table 3 corresponding to the nonzero bits of the byte.

Table 3.
Reduced lookup table for CRC-16.

Power of x (i)	$R_{g(x)}$ [x^i] (in hex)
16	A001
17	F001
18	D801
19	CC01
20	C601
21	C301
22	C181
23	C0C1

On-the-fly algorithm

Another approach that keeps the storage requirement low is the on-the-fly, or CRCF, algorithm.[3] In this algorithm, we compute each modified check bit (s_i') as a function of the original check bits (s_i's) and the sum of the original check bits and the input bits (t_i's). We can derive the required functional relationships by using the reduced lookup table, as shown in Table 4 for the CRC-16 polynomial. The first row in Table 4 corresponds to the original check bits, that is, $s(x)$ shifted eight positions to the right. The next eight rows correspond to the entries of Table 3 expressed in binary with the nonzero bits replaced by the appropriate symbolic notation $t_i, i = 0, 1, \ldots, 7$. Table 5 is a re-

Table 4.
Relationships leading to the on-the-fly algorithm.[3]

s_0'	s_1'	s_2'	s_3'	s_4'	s_5'	s_6'	s_7'	s_8'	s_9'	s_{10}'	s_{11}'	s_{12}'	s_{13}'	s_{14}'	s_{15}'
0	0	0	0	0	0	0	0	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
t_0	0	t_0	0	0	0	0	0	0	0	0	0	0	0	0	t_0
t_1	t_1	t_1	t_1	0	0	0	0	0	0	0	0	0	0	0	t_1
t_2	t_2	0	t_2	t_2	0	0	0	0	0	0	0	0	0	0	t_2
t_3	t_3	0	0	t_3	t_3	0	0	0	0	0	0	0	0	0	t_3
t_4	t_4	0	0	0	t_4	t_4	0	0	0	0	0	0	0	0	t_4
t_5	t_5	0	0	0	0	t_5	t_5	0	0	0	0	0	0	0	t_5
t_6	t_6	0	0	0	0	0	t_6	t_6	0	0	0	0	0	0	t_6
t_7	t_7	0	0	0	0	0	0	t_7	t_7	0	0	0	0	0	t_7

Table 5.
Rearranged form of Table 4.

s_0'	s_1'	s_2'	s_3'	s_4'	s_5'	s_6'	s_7'	s_8'	s_9'	s_{10}'	s_{11}'	s_{12}'	s_{13}'	s_{14}'	s_{15}'
0	0	0	0	0	0	0	0	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
t_0	t_1	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7						t_0
t_1	t_2	t_1	t_2	t_3	t_4	t_5	t_6	t_7							t_1
t_2	t_3														t_2
t_3	t_4														t_3
t_4	t_5														t_4
t_5	t_6														t_5
t_6	t_7														t_6
t_7															t_7

arranged form of Table 4 and is identical to Perez's Table 5[3] except for notational differences. We describe the on-the-fly algorithm here:

1) Initialize the CRC register to 0000 hex.

2) EXOR the input byte $(b_0, b_1, ..., b_7)$ with $(s_8, s_9, ..., s_{15})$ to form $(t_0, t_1, ..., t_7)$.

3) Shift the CRC register eight positions to the right.

4) Calculate the contribution to each s_i', $i = 0, 1, ..., 15$ from the t_i's using the relationships in Table 5.

Table 6.
Reduced lookup tables for some important CRC polynomials including Ethernet.

CRC-16		SDLC		CRC-16 Reverse		SDLC Reverse	
i	$R_{g(x)}[x^i]$	i	$R_{g(x)}[x^i]$	i	$R_{g(x)}[x^i]$	i	$R_{g(x)}[x^i]$
16	A001	16	8408	16	C002	16	8810
17	F001	17	4204	17	6001	17	4408
18	D801	18	2102	18	F002	18	2204
19	CC01	19	1081	19	7801	19	1102
20	C601	20	8C48	20	FC02	20	0881
21	C301	21	4624	21	7E01	21	8C50
22	C181	22	2312	22	FF02	22	4628
23	C0C1	23	1189	23	7F81	23	2314
24	C061	24	8CCC	24	FFC2	24	118A
25	C031	25	4666	25	7FE1	25	08C5
26	C019	26	2333	26	FFF2	26	8C72
27	C00D	27	9591	27	7FF9	27	4639
28	C007	28	CEC0	28	FFFE	28	AB0C
29	C002	29	6760	29	7FFF	29	5586
30	6001	30	33B0	30	FFFD	30	2AC3
31	9001	31	19D8	31	BFFC	31	9D71

Ethernet							
i	$R_{g(x)}[x^i]$	i	$R_{g(x)}[x^i]$	i	$R_{g(x)}[x^i]$	i	$R_{g(x)}[x^i]$
32	EDB88320	40	3B83984B	48	E1351B80	56	ED59B63B
33	76DC4190	41	F0794F05	49	709A8DC0	57	9B14583D
34	3B6E20C8	42	958424A2	50	384D46E0	58	A032AF3E
35	1DB71064	43	4AC21251	51	1C26A370	59	5019579F
36	0EDB8832	44	C8D98A08	52	0E1351B8	60	C5B428EF
37	076DC419	45	646CC504	53	0709A8DC	61	8F629757
38	EE0E612C	46	32366282	54	0384D46E	62	AA09C88B
39	77073096	47	191B3141	55	01C26A37	63	B8BC6765

EXOR the CRC register with the calculated 16-bit value.

5) Repeat steps 2 to 4 until you reach the end of the message.

Wordwise algorithms

The software algorithms we have discussed so far, except for the bitwise algorithm, handle data a byte at a time. These algorithms can easily be modified to handle larger units of data provided the size of such a unit does not exceed the degree of the generator polynomial. In Table 6 we list reduced lookup tables for some important generator polynomials. The program shown in the accompanying box generated these tables.

For each $g(x)$, we provide the values of $R_{g(x)}[x^i]$ for $n - k \leq i < 2(n - k)$. Such tables are useful in generating the necessary information for implementing the different software algorithms with any unit of data the size of which does not exceed the degree of the generator polynomial. For example, consider the CRC-16 polynomial with a word as the unit of data. The corresponding reduced lookup table can be used to generate a regular lookup table with 65,536 entries necessary to implement the table lookup algorithm. It can be used directly in a wordwise reduced table lookup, or CRCR(W), algorithm. It can also be used to generate the necessary functional relationships for implementing the on-the-fly algorithm. The descriptions of the different algorithms using a word as the unit of data undergo minor modifications. We describe the wordwise reduced table lookup algorithm, for example, as follows:

1) Initialize the CRC register to 0000 hex.

2) EXOR the input word $(b_0, b_1, \ldots, b_{15})$ with $(s_0, s_1, \ldots, s_{15})$ to form $(t_0, t_1, \ldots, t_{15})$.

3) Shift the CRC register 16 positions to the right.

4) For $i = 0, 1, \ldots, 15$ if $t_i = 1$, look up the corresponding 16-bit value from the table and EXOR the CRC register with it.

5) Repeat steps 2 to 4 until you reach the end of the message.

We compared the different software algorithms in terms of their speeds and storage requirements. Table 7 lists the results. These algorithms are the bitwise CRCB, the table lookup CRCT, the on-the-fly CRCF, and the bytewise and wordwise reduced table lookup CRCR(B) and CRCR(W). The generator polynomial used in the comparison is the CRC-16 polynomial. The accompanying box lists the five programs implementing the different algorithms. The programs are written in 8086 assembly language as routines to be called from a main C program. We used an Intel single-board computer (iSBC 86/12) operating at 5-MHz clock speed in executing the different programs.

Table generation

LISTING 1

```
/*****************************************************************
*                                                                *
* This program interactively generates a reduced look-up table for any *
* given generator polynomial g(x) of degree 16 or 32. The information *
* about the polynomial is entered in two steps as follows:       *
*                                                                *
*              1)  Degree of the polynomial (16 or 32).          *
*              2)  Coefficients of the polynomial except that of *
*                  the highest degree term in hex notation (4 or *
*                  8 hex characters).                            *
*                                                                *
* For example, consider the CRC-16 polynomial with g(x) = 1 + x**2 + *
* x**15 + x**16.  When prompted, information about CRC-16 is entered *
* as follows:                                                    *
*              1)  Enter degree of the polynomial (16 or 32).    *
*                  16 <CR>                                       *
*              2)  Enter coefficients of the polynomial in hex.  *
*                  A001 <CR>                                     *
*                                                                *
* The value A001 is obtained from the coefficients of the CRC-16 *
* polynomial (except the highest degree term) arranged as:       *
*                                                                *
*              1.(x**0)+0.(x**1)+1.(x**2)+.....+1.(x**15)        *
*                                                                *
*                                                                *
* NOTE:  This program assumes 32 bit integers                    *
*                                                                *
*****************************************************************/

main()

{

 int     degree, coeff, remainder, temp, power;

/*  get information about the generator polynomial  */

 printf("Enter degree of the polynomial (16 or 32).\n");
 scanf("%d",&degree);
 while (degree != 32 && degree != 16)    {
        printf("Only 16 and 32 are allowed.\n");
        printf("Enter degree of the polynomial (16 or 32).\n");
        scanf("%d",&degree);
 }
 printf("Enter coefficients of the polynomial in hex.\n");
 scanf("%X", &coeff);

/*  compute and print the reduced look-up table  */

 printf("Power of x [=i]   R((x**i)/g(x))\n");
 printf("---------------   ---------------\n");
 remainder = coeff;
 for (power = degree; power < 2*degree; power++) {
        printf("      %d                %X\n", power, remainder);
        temp = remainder & 0x00000001;
        remainder = (remainder >> 1);
        if (temp == 1)
                remainder = remainder ^ coeff;
 }

}               /*  end main  */
```

The programs implementing the five CRC algorithms

```
;  This module contains 8086 assembly language routines implementing
;  different CRC algorithms for the CRC-16 polynomial. These routines
;  are to be invoked from a main C language program (not shown). This
;  module is to be assembled using the Aztec C86 software package.

        largecode
        assume  cs:codeseg, ds:dataseg

dataseg segment para    public  'data'

table1  dw      0000h                   ;the 256 entry look-up table
;                 .                     ;for CRC-16 goes here
;                 .

table2  dw      0A001h                  ;the 16 entry reduced look-up
;                 .                     ;table for CRC-16 goes here
;                 .

dataseg ende

codeseg segment para    public  'code'


;                           LISTING 2

;                       Bit-wise Algorithm

;*************************************************************
;*                                                           *
;*  crcb (buff_addr,length)                                  *
;*                                                           *
;*  This routine computes the check bits for any given message *
;*  using the bit-wise algorithm, i.e., software simulation of *
;*  the LFSR circuit hardware implementation.                *
;*                                                           *
;*  Called from "C" program as :  crcb(buff_addr,length);    *
;*                                                           *
;*  Input     :     buff_addr  -  message buffer address     *
;*                  length     -  message length (words)     *
;*                                                           *
;*  Output    :     check bits -  returned in ax             *
;*                                                           *
;*************************************************************

        public  crcb_
crcb_   proc    far

        push    bp              ;standard prolog
        mov     bp,sp
        push    es
        push    di
        push    si
        push    cx
        push    dx

        lss     di,6[bp]        ;load buffer address in ss:di
        mov     si,10[bp]       ;load message length in si

        xor     ax,ax           ;initialize CRC register
crcb0:  mov     dx,es:[di]      ;fetch a message word
        add     di,2            ;bump message pointer
        mov     cx,16           ;load word length (16 bits) in cx
crcb1:  clc                     ;prepare for a shift
        rcr     ax,1            ;shift CRC register to right
        jc      crcb2           ;check lsb of CRC register
        rcr     dx,1            ;check lsb of message word
        jnc     crcb4           ;0-0 - no action
        jmp     crcb3           ;0-1 - modify CRC registsr
crcb2:  rcr     dx,1            ;check lsb of message word
        jc      crcb4           ;1-1 - no action
crcb3:  xor     ax,0A001h       ;1-0 - modify CRC register
crcb4:  loop    crcb1           ;keep looping till end of word
        dec     si              ;
        jnz     crcb0           ;keep looping till end of message
        pop     dx              ;standard epilog
        pop     cx
        pop     si
        pop     di
        pop     ss
        pop     bp
        ret
crcb_   endp

;                           LISTING 3

;                   Table Look-up Algorithm
;*************************************************************
;*                                                           *
;*  crct (buff_addr,length)                                  *
;*                                                           *
;*  This routine computes the check bits for any given message *
;*  using ths tabls look-up algorithm.                       *
;*                                                           *
;*  Called from "C" program as:   crct(buff_addr,lsngth);    *
;*                                                           *
;*  Input     :     buff_addr  -  messags buffer address     *
;*                  length     -  message length (bytes)     *
;*                                                           *
;*  Output    :     check bits -  rsturned in ax             *
;*                                                           *
;*  NOTE:  The 256 sntry (512 bytes) look-up table starts at *
;*         label "table1" within the data segment.           *
;*                                                           *
;*************************************************************

        public  crct_
crct_   proc    far

        push    bp              ;standard prolog
        mov     bp,sp
        push    ss
        push    di
        push    bx
        push    cx

        lss     di,6[bp]        ;load buffer address in es:di
        mov     cx,10[bp]       ;load meseage length in cx

        xor     ax,ax           ;initialize CRC rsgieter
crct0:  mov     bl,es:[di]      ;fetch a message byte
        inc     di              ;bump message pointer
        xor     bl,al           ;XOR messags byts with low CRC byte
        mov     al,ah           ;shift CRC registsr 8 positions to
        xor     ah,ah           ; ths right
        xor     bh,bh           ;prepare bx for indexing
        shl     bx,1            ;multiply bx by 2
        xor     ax,table1[bx]   ;XOR CRC register with table sntry
        loop    crct0           ;keep looping till snd of messags

        pop     cx              ;etandard epilag
        pop     bx
        pop     di
        pop     ss
        pop     bp
        rst
crct_   sndp
```

```
;                          LISTING 4

;                    On-the-fly Algorithm

;*****************************************************************
;*                                                               *
;*  crcf (buff_oddr,length)                                      *
;*                                                               *
;*  This routine computes the check bits for any given meseage   *
;*  ueing the on-the-fly algorithm [3].                          *
;*                                                               *
;*  Called from "C" program as :  crcf(buff_addr,length);        *
;*                                                               *
;*  Input     :     buff_addr  -  meeeage buffer address         *
;*                  length     -  message length (bytes)         *
;*                                                               *
;*  Output    :     check bits -  returned in ax                 *
;*                                                               *
;*  NOTE:  See [3] for o description of the logic and the        *
;*         eymbolic nototions used in this routine.              *
;*                                                               *
;*****************************************************************

        public  crcf_
crcf_   proc    far

        push    bp              ;etandard prolog
        mov     bp,sp
        push    es
        push    di,
        push    si
        pueh    cx
        pueh    dx

        les     di,6[bp]        ;load buffer addrese in es:di
        mov     ei,10[bp]       ;load meseage length in si

        xor     ax,ax           ;initialize CRC register
crcf0:  mov     dl,ee:[di]      ;fetch a meseage byte
        inc     di              ;bump meseage pointer
        xor     dl,al           ;xor meseage byte with low CRC byte
        mov     dh,dl           ; to form x and save it
        add     dh,dh           ;compute x8 & xx7 as carry and parity
        puehf                   ;save the flogs
        xor     dl,dh           ;compute R14 through R7
        xor     dh,dh           ;make xx7 and xx8 equol to 0
        popf                    ;reetore the flage
        jpe     crcf1           ;if xx7 octually is 1, moke xx7 and
        mov     dh,011b         ; xx8 equal to 1
crcf1:  jnc     crcf2           ;if x8 equale 1, then
        xor     dh,010b         ; complement xx8
crcf2:  mov     al,dl           ;load R14 through R7 in low CRC byte
        mov     ch,ah           ;eave high CRC byte
        mov     ah,dh           ;load R16 and R15 in high CRC byte
        ehr     dh,1            ;make xx8 the lsb
        mov     cl,6            ;shift CRC register six positions
        shl     ax,cl           ; to left
        or      al,dh           ;CRC register has R16 through R1
        xor     al,ch           ;xor with high CRC byte
        dec     si              ;
        jnz     crcf0           ;keep looping till end of message

        pop     dx              ;etandard epilog
        pop     cx
        pop     si
        pop     di
        pop     es
        pop     bp
        ret

crcf_   endp
```

```
;                          LISTING 5

;                Reduced Table Look-up Algorithm
;                        (Byte-wise)

;*****************************************************************
;*                                                               *
;*  crcrb (buff_addr,length)                                     *
;*                                                               *
;*  This routine computes the check bits for any given messoge   *
;*  using the byte-wise reduced toble look-up algorithm.         *
;*                                                               *
;*  Called from "C" progrom os :  crcrb(buff_addr,length);       *
;*                                                               *
;*  Input     :     buff_addr  -  message buffer address         *
;*                  length     -  message length (bytes)         *
;*                                                               *
;*  Output    :     check bite -  returned in ax                 *
;*                                                               *
;*  NOTE:  The 16 entry (32 bytes) reduced look-up table         *
;*         starte at label "toble2" within the doto segment.     *
;*         This routine uses only the firet eight entriee of     *
;*         the table.                                            *
;*                                                               *
;*****************************************************************

        public  crcrb_
crcrb_  proc    for

        push    bp              ;standard prolog
        mov     bp,ep
        push    es
        push    di
        pueh    ei
        push    bx
        push    cx
        push    dx

        lee     di,6[bp]        ;load buffer address in ee:di
        mov     si,10[bp]       ;load message length in si

        xor     ax,ax           ;initialize CRC register
crcrb0: mov     dl,es:[di]      ;fetch a meeeage byte
        inc     di              ;bump message pointer
        xor     dl,al           ;xor message byte with low CRC byte
        mov     al,ah           ;ehift CRC register 8 poeitions to
        xor     ah,ah           ; the right
        mov     cx,8            ;load byte length (8 bits) in cx
        xor     bx,bx           ;lood table index in bx
crcrb1: rcl     dl,1            ;check the msb of dl
        jnc     crcrb2          ;0 - no action
        xor     ax,table2[bx]   ;1 - modify CRC register
crcrb2: add     bx,2            ;bump table index
        loop    crcrb1          ;keep looping till end of byte
        dec     ei              ;
        jnz     crcrb0          ;keep looping till end of meseage

        pop     dx              ;standard epilog
        pop     cx
        pop     bx
        pop     si
        pop     di
        pop     es
        pop     bp
        ret

crcrb_  endp
```

```
;                       LISTING 6

;              Reduced Toble Look-up Algorithm
;                       (Word-wise)


;*****************************************************************
;*                                                               *
;*  crcrw (buff_addr,length)                                     *
;*                                                               *
;*  This routine computes the check bits for ony given messoge   *
;*  using the word-wise reduced toble look-up olgorithm.         *
;*                                                               *
;*  Called from "C" program os :  crcrw(buff_oddr,length);       *
;*                                                               *
;*  Input     :     buff_oddr  -  messoge buffer oddress         *
;*                  length     -  meesoge length (words)         *
;*                                                               *
;*  Output    :     check bits -  returned in ox                 *
;*                                                               *
;*  NOTE:  The 16 entry (32 bytes) reduced look-up table         *
;*         starts ot lobel "table2" within the doto segment.     *
;*                                                               *
;*****************************************************************

        public  crcrw_
crcrw_  proc    far

        push    bp              ;stondord prolog
        mov     bp,sp
        push    es
        push    di
        push    si
        push    bx
        push    cx
        push    dx

        les     di,6[bp]        ;lood buffer oddress in es:di
        mov     si,10[bp]       ;lood messoge length in si

        xor     ox,ox           ;initiolize CRC register
crcrw0: mov     dx,es:[di]      ;fetch o messoge word
        odd     di,2            ;bump meesoge pointer
        xor     dx,ox           ;xor messoge word with CRC register
        xor     ox,ox           ;shift CRC register 16 positions to
                                ; the right
        mov     cx,16           ;lood word length (16 bits) in cx
        xor     bx,bx           ;load toble index in bx
crcrw1: rcl     dx,1            ;check the msb of dx
        jnc     crcrw2          ;0 - no oction
        xor     ox,toble2[bx]   ;1 - modify CRC register
crcrw2: odd     bx,2            ;bump table index
        loop    crcrw1          ;keep looping till end of word
        dec     si              ;
        jnz     crcrw0          ;keep looping till end of messoge

        pop     dx              ;stondord epilog
        pop     cx
        pop     bx
        pop     si
        pop     di
        pop     es
        pop     bp
        ret

crcrw_  endp


codeseg ends
```

Table 7.
Comparison of CRC algorithms.

Algorithm	Time (ms) (for 512 bytes)	Storage (bytes)
CRCB	61.2	56
CRCT	8.4	553
CRCF	18.8	67
CRCR(B)	53.3	72
CRCR(W)	50.9	88

In Table 7 we see the results of a comparison of the execution speeds of the different algorithms; we compared the times taken to encode a 512-byte long message. The bitwise algorithm is seen to be the slowest. In comparison, the reduced table lookup algorithms are about 20 percent faster. The on-the-fly and table lookup algorithms are faster by a factor of about 3 and 7. The table lookup algorithm requires the largest amount of storage, while the other algorithms have relatively moderate storage requirements. In terms of the flexibility to use different generator polynomials, all programs except the one implementing the on-the-fly algorithm require minimal recoding.

Acknowledgments

We wish to express our gratitude to Jim Davis of Iowa State University for his support and encouragement during the course of this work.

References

1. J.S. Whiting, "An Efficient Software Method for Implementing Polynomial Error Detection Codes," *Computer Design,* Mar. 1975, pp. 73-77.
2. R. Lee, "Cyclic Code Redundancy," *Digital Design,* July 1981, pp. 77-85.
3. A. Perez, "Byte-wise CRC Calculations," *IEEE Micro,* June 1983, pp. 40-50.
4. W.W. Peterson and D.T. Brown, "Cyclic Codes for Error Detection," *Proc. IRE,* Jan. 1961, pp. 228-235.
5. *ANSI/IEEE Std. 802.3-1985, "Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications,"* 1985.

Tenkasi V. Ramabadran is an assistant professor of electrical engineering and computer engineering at Iowa State University. His current research interests include data compression, error control coding, speech coding, and signal processing.

Ramabadran received his BE from Regional Engineering College, Trichy, India; his MTech from Indian Institute of Technology, Madras; and his MS and PhD from the University of Notre Dame. All were in electrical engineering.

Sunil S. Gaitonde is working for his PhD degree in electrical engineering and computer engineering at Iowa State University. His research interests include protocol design; local area network design and performance; and the integration of voice, data, and video on computer networks.

Gaitonde received a BTech degree in electrical engineering from the Indian Institute of Technology in Kharagpur and an MS in computer engineering from ISU.

Questions concerning this article can be addressed to Tenkasi Ramabadran, 332 Coover, Department of EE&CE, Iowa State University, Ames, Iowa 50011.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Interest Card.

Low 165 **Medium** 166 **High** 167

Department

MicroReview

Richard Mateosian
2919 Forest Avenue
Berkeley, CA 94705-1310
(415) 540-7745

True BASIC (True BASIC, Inc.)

John G. Kemeny and Thomas E. Kurtz invented the Basic programming language in the 1960s, but by 1978 the Basic that most people used was some variant of Bill Gates' Basic for the Intel 8080 microprocessor. This was the product that gave Microsoft its start, but interest in it has declined since the advent of IBM's PC and Apple's Macintosh. These computers have shifted the focus of personal computing toward spreadsheets, desktop publishing, communications, and databases and away from personal recreational programming.

A few years ago Kemeny and Kurtz decided that they wanted to regain control of Basic, which had drifted fairly far from their original conception, particularly in the area of machine independence. Original Basic had been truly machine independent, but the various incarnations of Microsoft Basic, each usually funded by a computer manufacturer, were aggressively machine dependent.

The choice of the name *True BASIC* for the new Kemeny and Kurtz product is a little disingenuous. The True Basic of 1988 is far beyond anything that Kemeny and Kurtz could have planned for Basic in 1965. While block-structured languages certainly existed in 1965, the whole concept of the GOTO-less "structured program"—foreshadowed in Martin Richards' Berkeley California programming language (BCPL), formalized by Dijkstra, and popularized in Wirth's Pascal—was years in the future. Modules, currently popular in Modula 2 and Ada, had not been formalized in 1965. Libraries, separate compilation, and dummy arguments in subroutine definitions were all well-known aspects of Fortran in 1965, but Kemeny and Kurtz made no attempt to incorporate them into Basic at that time. In fact, the only important features of True Basic that are true to the original intention of Basic are its simplicity and its machine independence. I'd prefer a name for the product that shows that Basic has changed with the times rather than one that suggests that Kemeny and Kurtz foresaw the future.

The name aside, True Basic is a wonderful programming language for 1988. It retains the simplicity of Basic, yet it corrects most of Basic's well-known flaws. In fact, if you consider the OPTION TYPO and LOCAL features, which allow reasonable scope to be attached to variable names, you might say that it corrects all of Basic's important flaws. It has genuine subroutines (totally abandoning the GOSUB/RETURN mechanism of original Basic) and it no longer uses line numbers (except in a restricted mode designed for compatibility with original Basic). It also has modern control structures, a sophisticated module structure, and much more.

Another feature of True Basic that echoes back to the early days is the set of matrix instructions. Kemeny and Kurtz seem to have had this bee in their bonnets for a long time, perhaps because of their interest in using the language for mathematics education. I didn't think much of the idea in 1967, and I still don't. Like some of the fancy instructions of the complex instruction set computer (CISC) architectures, True Basic's matrix instructions will be seldom-used baggage that every user of the language will be forced to carry.

True Basic is about as machine independent as it can be. This is possible because the language contains the fundamental elements of screens, windows, color, sound, and sequential and random-access files. These elements are then implemented differently on each kind of computer. For most programs, this is enough, but if specific machine features need to be accessed, this can be done through libraries, such as one for the Macintosh that facilitates access to the ROM-based Toolbox.

Libraries can be used to extend True Basic quite smoothly. Libraries can be loaded *in toto* with True Basic, in which case the functions in them become indistinguishable from those built into the language. Or, if only a few functions are needed from a library, that library can be loaded by program statements, in which case only the needed functions will be loaded. This uses less memory but puts the burden on the user of assuring that the libraries loaded and the functions within each library are in the proper order. True Basic makes a single pass over the specified libraries, loading only those functions for which a need has already been declared. If a subsequently loaded function needs to call a library function that was previously passed over, an error will result.

True Basic (the company) makes available a selection of libraries for specific purposes. The ones listed in the material they sent me are the *Scientific Graphics Toolkit*, the *Business Graphics Toolkit*, the *3-D Graphics Toolkit*, the *Mathematician's Toolkit*, and the *Runtime Package*. The *Runtime Package* is used to turn True Basic programs into freestanding applications for the given computer. The computers for which versions of True Basic and these libraries exist are the IBM PC and PS/2, the Apple Macintosh, the Commodore Amiga, and the Atari ST.

True Basic's machine independence is emphasized by the structure of the documentation. The principal document is the *True BASIC Reference Manual* by

Kemeny and Kurtz. The other important document for any given implementation of True Basic (say for the IBM PC) is the user's guide, which contains information specific to that implementation.

Additional documentation accompanies each library, and this can be either almost entirely generic, as in the case of the scientific packages, or entirely specific, as in the case of the runtime packages.

As I usually do, I began my evaluation of True Basic by reading the documentation. I read the user's guide first, and then the reference manual. I recommend this procedure to anyone preparing to use a new software package. Manuals are often informative (sometimes in ways the authors didn't intend), and these were no exception. They are clear, well organized, and adequately detailed. Furthermore, they reveal an intelligence and an understanding of issues that few manual authors are blessed with. Finally, they focus on ways to use the language effectively rather than on covering up its defects or touting it against the competition.

I didn't write any programs in True Basic, but I looked over and ran all of the sample programs they sent and tried the debugging features. I have a lot of experience with Basic, and True Basic seems to provide everything the Basic programmer needs. I recommend it highly.

Reader Service Number 4

Initial Computability, Algebraic Specifications, and Partial Algebras, Horst Reichel (Oxford University Press, New York, 1987, 221 pp; $55)

This is the second of "The International Series of Monographs on Computer Science," but it is difficult to see how the material in this book will ever have much application to the difficulties encountered in the programming of computers.

Reichel begins by quoting Hoare: "In the development of programs by stepwise refinement the programmer is encouraged to postpone the decision on representation of his data until after he has designed his algorithm and has expressed it as an abstract program operating on abstract data." Reichel sees in this a problem, namely, to make precise the concepts of abstract program and abstract data. This leads him into many-sorted algebras, small categories, and universal algebras, and from there he comes to the main focus of his book, partial algebras.

By now you've gotten the idea that this is a theoretical book, and this is true. Many examples are given to try to tie the theory of partial algebras to familiar concepts of computer science like lists, trees, and stacks, but these attempts are largely unsuccessful. The early examples—the ones I felt I ought to be able to understand—were marred by obvious typos, while the later examples were sufficiently incomprehensible that I often couldn't decide whether or not certain mysterious lines contained errors.

Perhaps other readers, better prepared than I am, will be able to make more sense out of this book. I hope so. For purposes of calibration, let me state that I have a PhD degree in mathematics from UC Berkeley and 27 years of experience in computer science.

Advanced Research in VLSI—Proceedings of the Fifth MIT Conference, March 1988, Jonathan Allen and F. Thomson Leighton, eds. (MIT Press, Cambridge, Mass., 1988, 380 pp; $45)

In my first column (April 1987) I reviewed a book from the MIT Press that was badly edited and full of typographical errors. This one, by contrast, is an excellent example of high-quality technical publishing. The conference in question was held in March 1988, and I received my review copy of the book in May, but despite this prompt publication, the papers are well written, the illustrations are clear, and the mathematics—some quite complex—is beautifully displayed. The only typos I've noticed are in the editors' preface. The astonishing fact is that the entire book was photographed directly from the authors' prepared copy. I have had considerable experience working with authors' prepared copy, and I have rarely received papers anywhere near as good as the average paper in this book. If the editors will let me in on their secret, I'll be eternally grateful.

The conference documented in this book was conducted in a single track, with 19 papers and nine "invited talks." All of the papers appear in the book, but only the titles and brief previews are given for most of the invited talks. I'm not sure why I'd want to read previews of talks I'm never going to hear, but little harm has been done by including them, and without them (and the blank pages around them), the book would be a little too thin to justify the price.

The 19 papers cover a wide range of topics, all generally related to the design of high-performance, very large scale integration circuits. They address architectures, timing issues, parallel computation, fault tolerance, layout, design verification, and CAD tools. There are theoretical analyses and practical discussions, with a slight bias toward the theoretical.

One of the easier papers to read, but extremely interesting nonetheless, is a retrospective on the design and testing of MIPS-X, the second generation of Stanford's MIPS machine, one of the early reduced instruction set computers. The paper discusses "the strengths and weaknesses of our tools, methodology and design environment." The authors give a good picture of how the project worked and how the team members interacted, how the tools were chosen, how they supported the activity, and where they fell short.

Another extremely interesting paper describes work done at the California Institute of Technology on a VLSI circuit for processing visual images. Phototransistors and analog processing elements have been included on a single die. This means that large numbers of signals can be routed simultaneously to many processing elements, allowing real-time computations that would not be possible in multichip configurations. This circuit has been used successfully in the laboratory to track the position of a bright spot moving against a dark background.

If you're interested in new developments in VLSI design, you'll find a lot of interesting material in this book.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Interest Card.

Low 177 Medium 178 High 179

Department

MicroStandards

Michael Smolin
Smolin & Associates
3428 Greer Road
Palo Alto, CA 94303

Project update

It is time for an update of microstandards working groups under the sponsorship of the Technical Committee for Microprocessors and Microcomputers (TCMM), administered by the Microprocessor Standards Committee (MSC). My last complete project update report was in the October 1987 issue of *IEEE Micro*. Table 1 lists each current project, its status, and its chair. Many projects already have culminated in adopted IEEE standards. Table 2 lists similar information for study groups.

New microstandards published

Two IEEE standards adopted in 1987 are now published and available:

- *IEEE Standard 1014-1987, Versatile Backplane Bus* (VMEbus); and
- *IEEE Standard 1101-1987, Mechanical Core Specifications for Microcomputers*. (Standard 1101 will also be incorporated in the publications of IEEE Standards 896.1, 1196, and 1296.)

Order standards from the IEEE Service Center, PO Box 1331, 445 Hoes Lane, Piscataway, NJ 08855-1331, phone (201) 562-5346 (for credit card or COD orders); or the Computer Society Press, PO Box 4699, Los Angeles, CA 90051, (800) CS-Books.

From the Standards Board

Actions taken at the June meeting of the IEEE Standards Board include:

- reaffirmation of *IEEE Standard 796, Microcomputer System Bus* (Multibus I);
- adoption of IEEE Standard 959, I/O Extension Bus (SBX); and
- adoption of IEEE Standard 1096, Multiplexed High-Performance Bus Structure (VSB).

The board also authorized a new standards development project: P1178, Scheme Programming Language. For information contact chair Christopher T. Haynes (see Table 1). Also authorized was P1177, Fastbus Standard Routines, sponsored by the Nuclear and Plasma Sciences Society. Contact chair David Gustavson, (415) 926-2863, ext. 2574, for information.

Sponsor ballots

P970, Advanced Backplane Bus, has completed its sponsor ballot and should be presented to the Standards Board in October. (Yes, a change of time and venue. See "Standards forum.")

IEEE Standard 695 (Trial-Use), MUFOM—Microprocessor Universal Format for Object Modules, will be reballoted for approval as a full-use standard when its balloting body is formed.

IEEE Standard 855 (Trial-Use), MOSI—Microprocessor Operating System Interface, is expected to be balloted soon for revision and full-use status.

Standards forum to be held in Reno

The IEEE is sponsoring the Transnational Forum on Standards in Reno, Nevada, October 17-21, 1988. Forum topics on October 19 will include legal implications of standards, indemnification concerns, standardizing emerging technologies, and transnationalization of standards.

Many working groups and other standards committees (including the IEEE Standards Board) plan meetings on October 17, 18, 20, and 21. This may be an ideal time and place for you to attend standards meetings—or to run one. To arrange for meeting space, contact forum chair Jay Forster: (408) 370-4202 (work) or (408) 268-2320 (home).

Table 1.
TCMM/MSC working groups.

	Project/Description/Status	Chair
P695	**MUFOM—Microprocessor Universal Format for Object Modules (rev. IEEE Std. 695, Trial-Use)** Common formats for system-relocatable software used with microprocessors to provide for relocatable linking of separate code blocks. Ready for sponsor ballot to recommend as full-status standard.	Tom Pittman, PO Box 1848, Manhattan, KS 66502; (913) 776-1848
P696	**Interface for Microsystem Components (S-100, rev. IEEE Std. 696)** Revision allows regulated 5.0 VDC on the backplane in lieu of voltage regulation on each board. Some mechanical improvements included. Final draft expected from working group this summer.	Rick Kalish, Compupro/Vaisyn Corp., 26538 Danti Ct., Hayward, CA 94545-3999; (415) 786-0909
P755	**Extending High-Level Language Implementations for Microprocessors (rev. IEEE Std. 755, Trial-Use)** Major purpose was to provide a standard means for HLLs, especially Pascal, to communicate directly with hardware without requiring an operating system. Extended Pascal now includes such functionality, so authorization for this project will be withdrawn.	
P855	**MOSI—Microprocessor Operating System Interface (rev. IEEE Std. 855, Trial-Use)** Defines a set of programmatic interfaces between microprocessor operating systems and software that uses operating system functions. Final draft of revision expected out of working group this summer.	Jim Mooney, West Virginia Univ., Dept. of Statistics and Computer Sciences, Morgantown, WV 26506; (304) 293-3607
P896.2	**Futurebus Firmware** Provides cache-coherency and message-passing formats and other firmware requirements of multiprocessor systems based on IEEE Std. 896.1. Final draft due from working group by Nov. 1988.	Paul Borrill, National Semiconductor Corp., 2900 Semiconductor Dr., MS 16-181, Santa Clara, CA 95051; (408) 721-7443
P949	**MIIT—Media Independent Information Transfer (rev. IEEE Std. 949, Trial-Use)** Establishes a standard preamble format and cross-references other standards to provide a set of formats, independent of medium, which permit convenient exchange of information. Revision adds subformats such as medical, map, audio, and video to original data and text formats. Irregular working group meetings. Draft due first quarter of 1989.	Bob Davis, Summit Computer Systems Inc., 22685 Summit Rd., Los Gatos, CA 95030; (408) 353-2706
P970	**Advanced Backplane Bus (Versabus)** Resolving negative votes and comments of sponsor ballot. Draft expected at next IEEE Standards Board meeting.	Doug Kraft, Motorola, 11404 June Dr., Austin, TX 78753; (512) 837-4462
P996	**Extended Personal Computer Backplane Bus (PC Bus)** Project has developed standardized version of 16-bit PC Bus. Further, working group recommends IEEE Std. 1196—Nubus as 32-bit extension. Draft near completion and ready for cleanup and circulation to working group members. Final draft expected by end of 1988. Next meetings scheduled for Sept. 12 and Nov. 14.	Gordon Force (acting), Force Technology Corp., 4340 Almaden Expwy., Suite 240, San Jose, CA 95118; (408) 941-6065
P1132	**Versatile Serial Bus (VMS)** Concomitant to IEEE 1014 (VMEbus). IEEE and IEC standards for VMS are intended to be identical. IEEE and IEC are working to complete the standard.	Ad Willemse, Nederlandse Philips Bedrijven B.V., Bldg.TQ III-1-59, 5600 MD Eindhoven, The Netherlands; (040) 785-714
P1134	**Personal Computer Basic I/O Software** Project withdrawn for lack of interest.	
P1141	**Forth: A Microcomputer Language Standard** Aimed at making Forth fully transportable and extending it to 32-bit processors. Development of the standard is taking place in X3J14, with P1141 members representing the IEEE in that committee. Draft development is ongoing.	Guy Kelly, Practoserve, 10951 Sorrento Valley Rd., San Diego, CA 92121; (619) 454-1307

Table 1—continued.

P1151	**Modula 2, A Modular High-Level Programming Language** Group serves as the US TAG to the ISO working group developing the international standard for Modula 2. Development of the draft is ongoing in the ISO subcommittee.	Randy Bush, OSI, 6915 SW Macadam Ave., Portland, OR 97219; (503) 245-2202
P1152	**Smalltalk, An Object-Oriented Programming Language and Environment** Purpose is to define a standard that promotes the language, supports transport of developed programs, and minimizes spread of dialect variations. First two meetings set, in conjunction with ECOOP (Aug. 1988, Oslo, Norway) and OOPSLA (Sept. 1988, San Diego, CA).	L. Peter Deutsch, Park Place Systems Inc., PO Box 60264, Palo Alto, CA 94306, (415) 859-1023
P1154	**PILOT, A Program Instruction Learning or Teaching Language** Standardizing PILOT will improve code portability, minimize dialects, and promote sharing of courseware across teaching institutions. Working group committee meeting and developing draft.	John Starkweather, UCSF, 401 Parnassus Ave., Box 14C, San Francisco, CA 94143; (415) 476-7464
P1155	**A High-Speed Backplane Instrumentation Bus (Ibus)** Aims at a standard for a high-performance instrumentation bus for use with a variety of high-performance backplanes. Expected to support card-level instrumentation in military and industrial test and instrumentation systems. Many members of the Instrument and Measurement Society participate in this effort. Working group has selected VXIbus as starting point for its work. A draft is expected out of committee during the first quarter of 1989.	Marlyn Huckeby, Tektronix, 5770 Ruffin Rd., San Diego, CA 92123; (619) 292-1287; and Bill Maciejewski, Emerson Electric Co., Electronics and Space Division, 8100 W. Florissant, St. Louis, MO 63136; (314) 553-4628
P1156	**Connectors and Mechanical Packaging for High-Reliability Bus Structures** Intended to be the mechanical complement to P1496 Rugged Bus, but might well find use in implementing other buses in a harsh environment. Draft under development. Final draft expected March 1989.	Paul Cook, Ameritech Services Inc., 1900 E. Gulf Rd., Schaumberg, IL 60173; (312) 519-7643
P1178	**Scheme Programming Language** Scheme and Common Lisp are distinct members of the Lisp family of programming languages. Common Lisp is being defined by X3J13, while Scheme is being defined in P1178, based on the MIT RnRS Memo. New project—PAR approved by the IEEE Standards Board, June 1988.	Chris Haynes, Computer Science Dept., Indiana Univ., Bloomington, IN 47405; (812) 335-6486
P1394	**High-Performance Serial Backplane Bus** To be defined independent of the parallel backplane bus. Both intra- and intercrate capabilities are included. The 1394 will work with the 896, 1196, 1014, 1296, and other 32-bit buses. Ongoing working group meetings. Final draft due the fourth quarter of 1988.	Michael Teener, Apple Computer, MS/5-F, 10475 Bandley Dr., Cupertino, CA 95014; (408) 973-3521
P1395 **P1396**	**Physical I/O Configurations for Communication Switching Systems**, and **A Communication Bus for Hybrid Switching Applications (TDM bus)** A TDM bus for the telephone industry for PBX and central office equipment. Target is Sonet compatibility to T3 interfaces worldwide at 155 Mbits/second. Both projects meeting regularly.	Gary Nelson, Ameritech Services., Applied Technology and Development, 1900 E. Gulf Rd., Schaumberg, IL 60173; (312) 519-7620
P1496	**Rugged Bus: A Very High Reliability Bus Structure** Defines a bus structure for rugged military and industrial use. Concomitant project 1156 defines connectors and other mechanical structures. Strong SAE participation. Draft under development. Final draft expected by end of 1988.	Paul Cook, Ameritech Services Inc., 1900 E. Gulf Rd., Schaumberg, IL 60173; (312) 519-7643

Terms

ISO = International Standards Organization
OOPSLA = Conference on Object-Oriented Programming: Systems, Languages, and Applications
PAR = project authorization request
SAE = Society of Automotive Engineers
Sonet = Synchronous Optical Network
TAG = Technical Advisory Group

Table 2
Study groups.

	Project/Description/Status	Chair
1196.2	**High-Level Bus Conventions** Defines and coordinates use of status and register allocations for Nubus-like, self-configurable systems; provides standard way to identify a configuration. New proposed project. PAR will be submitted if significant interest develops.	George White, Corollary Inc., PO Box 18977, Irvine, CA 92713; (714) 250-4040
	Study Group on Logo Language Standard Project will standardize a proper subset of the Logo language. Currently a study group. Some disagreement over exact scope of project being resolved. Article on this project for TCMM newsletter will initiate formal PAR submittal.	Brian Harvey, 2634 Virginia Ct., #25, Berkeley, CA 94709; (415) 548-5228
	Study Group on Language for Data Manipulation Intended to standardize the language used in microprocessor database systems. Article for TCMM newsletter expected shortly, which will trigger formal PAR submittal.	Martin Rienhart, 233 Broadway, Suite 869, New York, NY 10279; (212) 406-7026
	Study Group on System Architecture (SGSA) Ongoing study group is looking at changes and descriptions of system architectures. Group formulates US response to actions by JTC1/SC47B Working Group 4 on System Architecture.	Gerry Laws, Texas Instruments, PO Box 2909, MS/2091, Austin, TX 78759; (512) 250-7640
	Study Group on Superbus (SGSB) The name is being changed to Scalable Coherent Interface (SCI). A proactive development for next-generation intercard communication structures at up to 1000-Mbyte/second transfers. Project scope and purpose being refined. Article for TCMM newsletter due, which will trigger PAR submittal.	Paul Sweazey, National Semiconductor, MS/D3678, PO Box 58090, Santa Clara, CA 95052-8090; (408) 721-5860
	Study Group for Relational Data Models (SGRDM) Studying feasibility of standardizing definitions used in microprocessor database systems. There now is no rigorous formal definition for describing a database system or against which a database system may be measured. If interest and support are sufficient, a PAR will be submitted to create a formal working group.	Jim Flournoy, Procedamus, 8655 Landis View, Rosemead, CA 91770; (213) 402-1610
	Study Group for Block Structured Tape Formats (SGBSTF) Goal is to standardize a method of building a block structure on helical scan tapes while permitting block replacement. This will let large data storage tapes behave like slow disks (transparently to the software). Group defining details of scope and purpose.	Dave Allen, Grassroots, 11065 W. 75th St., Aubry, KS 66062; (913) 681-3001.
	Study Group for Graphic Language Interfaces (SGGLI) Investigating feasibility, scope, and purpose of project.	Robert Stewart, Stewart Research Enterprises, 1658 Belvoir Dr., Los Altos, CA 94022; (415) 941-6699 (home)

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Interest Card.

Low 174 **Medium** 175 **High** 176

Department

MicroNews

MicroNews features information of interest to professionals in the microcomputer/microprocessor industry. Send information for inclusion in MicroNews one month before cover date to Managing Editor, IEEE Micro, *10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578.*

Tiny computer promises massive speeds

A microelectronics system small enough to fit into the palm of your hand and capable of 10 billion operations per second could grace your future.

Under development at Hughes Aircraft Company for the US Air Force's Rome Air Development Center is just such a computer. Initially, the experimental machine will process images and signals of complex radar information on space-based missions. Called a 3D computer because of both its physical structure and the way it performs operations (moving signals in three dimensions instead of two), this speedy processor radically departs from conventional computer design. Plans innovate a computer built almost completely from active silicon circuitry.

Physically, a number of IC wafers are stacked like record disks in a jukebox. Each wafer contains a large array of tiny processors, or work cells, that are replicated across its surface. Each work cell connects horizontally to its four neighboring cells. In addition, "microbridges" provide vertical processor-to-processor contact between wafers, and feedthroughs enable signals to pass through each wafer to adjacent ones. This combination of microbridges and feedthroughs permits the distribution of signals to all other cells in a vertical column. The computer's signals thus move in three dimensions and allow an extremely high degree of parallel processing for increased speed.

So far, laboratory tests have demonstrated a 3D computer structure using a five-wafer stack. Each wafer contains an array of 32 × 32 cells. The recent $5-million contract calls for evaluation of the design for computational throughput, fault tolerance, and operation under extreme mechanical and radiation environments. The main objective is a scaled-up, intermediate version that extends basic 3D technology to larger array sizes.

Other goals include developing more efficient wafer types, developing new software, and integrating hardware and software technologies to form a functioning 3D computer.

Hughes' major long-range goal is producing image-processing computers that can run at 100 billion operations per second in a package slightly larger than a tuna-fish can and consume less than 100 watts of power. In addition, the company hopes to apply the packaging technology of feedthroughs, interconnects, and wafer stacking to other IC designs such as extremely dense memories.

Deskpro 386/20 outperforms PS/2 Model 80

The Compaq Deskpro 386/20 produced the highest Khornerstone rating in a recent benchmark comparison of selected 80386-based computers. Representatives of Workstation Laboratories, the benchmark's developer, credit the Compaq's use of cache memory with the superior performance.

Table 1 lists the total Khornerstone ratings for tested machines and for individual CPU-intensive tests.

The Khornerstone is a balanced value of time measurements generated by 22 benchmarks. It represents a system's CPU, I/O, and floating-point performance.

Contact Phil Magney, ARS/Workstation Laboratories, 8111 LBJ Freeway, LB 156, Dallas, TX 75251; (214) 644-1733, for more information. Full results and source code are available.

Table 1.
Benchmark performances of 80386-based computers.

Microcomputer*	Microprocessor	CPU Khornerstones	Total Khornerstones
Compaq Deskpro 386/20	80386 (20 MHz)	1345	3927**
IBM PS/2 Mod 80	80386 (20 MHz)	987	2958**
Altos 386 Series 2000	80386 (16 MHz)	1250	3038
Apollo DN4000	68020 (25 MHz)	1472	5008
Sun 3/160	68020 (16.7 MHz)	792	3611
Sun 3/260	68020 (25 MHz)	1200	6767
Tektronix 4317	68020 (16 MHz)	995	3475

* All systems tested with floating-point chip or floating-point accelerator.
** Tested under MS/DOS; all others tested under Unix.

Organic computers a possibility?

Researchers continue to be intrigued with the possibility of building a computer with growth processes. One study has even produced superconducting wires through crystal growth.

Molecular electronics. Syracuse University professor of chemistry and director of its Center for Molecular Electronics Robert Birge hopes to use organic molecules to build an extremely precise CPU. With a complexity superior to that possible with semiconductor technology, the future computer would include an optically coupled molecular electronic device made of organic compounds bonded to a quartz plate.

According to Birge, MEDs, because of their size, have the potential to carry out logic functions and other information processing up to a thousand times faster than current supercomputers. This would let scientists conduct simulation experiments of complex phenomena in real time.

To overcome the fact that small molecules are difficult to hook up to necessary I/O wires, Birge wants to stimulate and interrogate molecules in one of two ways. He wants to use lasers or extremely high frequency millimeter (radio) waves in excess of 25 billion cycles per second. He has received a $300,000, three-year grant from the US Office of Naval Research to support his work.

Birge says that he is particularly interested in bacteriorhodopsin, a bacteria found in San Francisco salt marshes. The bacteria's purplish-red protein pigment absorbs a photon of light, stores it, then—through amplification—converts it into energy for the bacteria. "To date, nobody has found anything that can switch with the speed of bacteriorhodopsin," says Birge.

Birge feels he is in a race with his principal competitors, the Japanese, who are currently conducting most of the research in this field.

Epitaxial growth. Another growth process called selective epitaxial growth, or SEG, potentially could produce high-quality bipolar transistors that would be faster and denser than anything available through current technology. Mike Liu, principal engineering fellow at Honeywell's Solid State Electronics Laboratory, plans to make devices with this SEG technique for applications beyond the limits of current CMOS (complementary metal oxide semiconductor) and bi-CMOS technology.

Liu accomplishes SEG by first growing oxide on a silicon wafer and then cutting windows in the oxide using a photolithic process. With a plasma, he etches the oxide within the pattern. New silicon is then grown in the windows that have already been etched out, building either bipolar or MOS transistors.

The new silicon can also be grown up over the top of the silicon dioxide regions and etched back to produce a very thin silicon-on-insulator (SOI) structure. SOI can be used to make oxide-isolated semiconductors including CMOS, bipolar transistors, and JFET (junction field-effect transistors) for use with high levels of radiation and other hostile environments.

Liu says, "This process produces higher quality material than other methods to create oxide-isolated devices such as SIMOX (separation by implementation of oxygen) or SOS (silicon on sapphire)."

The key to SEG appears to be reduced pressure in the operation. Demonstrations at 50 torr (760 torr equals one atmosphere) and 900 degrees Centigrade have been successful at Honeywell.

Crystal growth. Meanwhile, Stanford researchers funded by the National Science Foundation have produced superconducting fibers capable of carrying large quantities of electrical current. A very clean, tightly focused laser heat source permitted steady-state, rapid, continuous production with no special containers or furnaces.

The laser-heated pedestal growth, or LHPG, method produces short wires that carry more than 30,000 amperes of electricity per square centimeter at 4 degrees Kelvin. The wires are a combination of bismuth, calcium, strontium, copper, and oxygen.

Researchers, led by Don Gazit, David Fork, and Theodore Geballe, mixed together appropriate amounts in the form of metal oxides or carbonates. They then finely ground, pressed into a pellet, and sintered the ingredients for 15 hours at 780 degrees C. Next, they ground, repressed, and again sintered the pellet. Finally, they cut small rectangular rods from the pellet and used them as the source material and for seeds to produce a crystal. (See Figure 1.)

A laser beam melts the top of the source rod, and a seed crystal appears in the melt. The fiber grows when scientists withdraw the seed at a controlled rate.

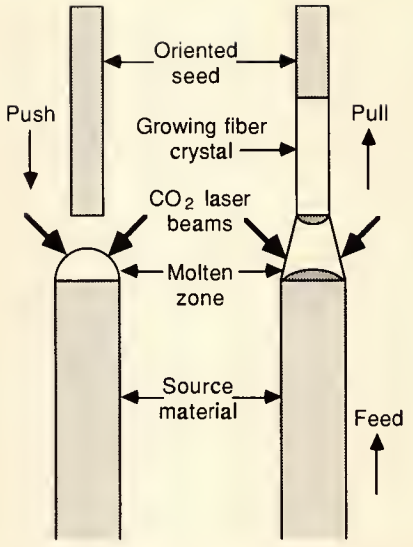


Figure 1. Producing superconducting fibers.

The LHPG method grows crystals, especially oxides, with diameters smaller than a human hair. Stanford researchers grew the superconducting fibers at a rate of 1.5 to 50 millimeters per hour.

Did you ever think you'd like a robot?

Microcomputers seem to go almost everywhere and do almost everything these days. Take your local hospital's robot, for instance. It is very likely activated and controlled by micros. Perhaps a quadriplegic patient needs help with simple tasks—like moving, eating, or picking up a telephone. Robotic health care technology might very well be used to "nurse" that patient.

Right now micro-controlled robots assist not only in patient recuperation but also in laundry areas and laboratories, where they're especially appreciated because they serve patients with contagious diseases. They also save the human wear and tear associated with lifting, transporting, and stacking large quantities of supplies.

Queen Street Mental Hospital Centre in Toronto, Ontario, uses individually controlled robotic carts to deliver food, trays, dishes, stored supplies, laundry, and pharmaceuticals from locked cabinets. They carry a maximum of 1,000 pounds and travel up to one mile an hour.

The University of Michigan Hospital at Ann Arbor makes use of a similar system. Each robot carries 1,500 pounds and glides under and lifts up to 750 pounds of medical and surgical supplies. This system costs $5 million and services an area measuring 1.3 million square feet.

Even surgeons acknowledge the need for using robots. For example, a robot arm used in Memorial Hospital Medical Center, Long Beach, California, guides surgeons during brain tumor biopsies. The arm—because of its accuracy—lets the surgeon drill smaller holes in a patient's skull. This improvement in turn shortens surgery time (90 minutes instead of four or five hours) and thus reduces the amount of anesthesia for the patient. The hospital also hopes to use its robot to insert stimulating electrodes and radioactive pellets into the brain and guide drainage accesses for abscesses.

Approximately 400 laboratory system robots are in use—and they pay for themselves in 11 to 18 months. These robots perform automated titration, sample preparation, enzyme-linked radioimmunoassay, and related tasks.

Demand for robots will grow 50 percent a year, according to Roberta J. Minifie of the University of South Florida. In a paper delivered earlier this year at The Institute of Management Sciences National Conference, Minifie also reported that robotic use is in its infancy, that Japan is the leading user of robotic technology, and that the United States produces 90 percent of the world's robotic equipment.

But, back to hospital care. According to Minifie's research, robots can help hospitals control their labor costs (approximately 50 percent of hospital costs are labor related) by releasing humans so they can offer more tender, loving care to patients. Aren't robots nice?

Computer history, Rev. 1

Was ENIAC the world's first electronic computer as we've always thought? Or, was it just a spin-off?

Computer historians at the University of Michigan, Ann Arbor, contend that another electronic computer predated ENIAC by four years.

"The First Electronic Computer: The Atanasoff Story," traces the history of computers and supports its ENIAC position with a 1973 federal court ruling upholding that Atanasoff's computer predated ENIAC. Authors Alice R. and Arthur W. Burks point out that John Vincent Atanasoff completed the world's first electronic computer in 1942. The special-purpose digital machine proved the feasibility of electronic computation and led directly to ENIAC, which was unveiled in 1946 to compute firing tables automatically.

Atanasoff developed his computer in the late 1930s with graduate student Clifford Berry while at Iowa State University.

The authors say they wrote the book in part because the court ruling in Atanasoff's favor has been ignored by the computing community and by historians. They want to set the record straight.

Alice Burks has been a researcher for the U-M Department of Electrical Engineering and Computer Science. Arthur Burks, professor emeritus in the same department, worked on ENIAC. He also worked with John von Neumann and Herman Goldstine to develop the logical design of an electronic digital computer with electronically stored programming. This basic design became the prototype for many computers built by universities, government research units, and IBM.

ESDI standard open for comment

ANSI's X3 committee invites the public to review and comment on the draft proposed American National Standard, X3.170-198x. Comments will be accepted until October 3, 1988.

This document defines the Enhanced Small Device Interface specification. ESDI supports magnetic and optical disks with a like physical interface, the same command/response protocol, and similar commands. The interface handles data rates from 0 to 24M bits per second, depending on the driver/receiver class used.

Copies of the draft standard cost $30 ($39 for international orders). Call Global Engineering Documents, Inc., at (800) 854-7179 if you'd like a copy.

Current Literature

An updated version of the *Guide to the Use of Standards* presents guidelines for the development of International Functional Standards in information technology. Revision 3, compiled by SPAG (the Standards Promotion Application Group), defines the ways in which

MicroBits

The American National Standards Institute announced the newest member of its family of **intelligent peripheral interface** standards, the ANSI X3.147-1988. The approved document isolates the CPU's hardware and software from changes by providing a "function-generic" command set to allow the connection of various peripherals.

Oracle Corporation, Toshiba, Altos, and **Priam** became the first computer industry sponsors of an Indianapolis 500 racing team this spring. The Randy Lewis Lola-Cosworth racing car is a product of computer-aided design systems. The team uses a computer for timing and scoring, and an onboard computer to monitor the engine and suspension.

Want up-to-the-minute information on all **ISDN** field trials and implementations on a global scale? Information Gatekeepers Inc. in Boston offers a free reference guide. Just fax your request to (617) 734-8562.

NuGroup is a **NuBus** Manufacturers and Users Group organized this April to target the five-year-old NuBus into 32-bit mainstream applications. NuGroup plans to show that NuBus is not only a good choice for personal computers but also an upgrade path for current STD and Multibus users. Call Anne Weber at (714) 669-1201 for more information.

IEEE's Educational Activities Department in New Jersey wants your help—if you're an expert in 16-bit microprocessors. A series of continuing education packages (the Individual Learning Program) includes an MPU study package. Expert reviewers/consultants receive a $250 honorarium, free series subscription, and free publicity.

information technology functions can be realized and aligns with the results achieved by the European standardization effort led by CEN/CENELEC/CEPT.

North-Holland, PO Box 1991, 1000 BZ Amsterdam, The Netherlands or Elsevier Science Publishing Co., PO Box 1663, Grand Central Station, New York, NY 10163; Dfl. 150 or $79.

Graeme Publishing offers both a newsletter and a primer report about neural networks. *Synapse Connection* is a monthly newsletter covering such topics as the "intelligent memory chip," which appears in the April issue.

Understanding Neural Networks: A Primer provides a 70-page overview of the field for potential users, technical managers, and investors, complete with illustrated descriptions.

Graeme Publishing, Ten Northern Boulevard, Amherst, NH 03031; (603) 886-8221; $295 a year (newsletter); $95 (report).

If you want to become a consultant, where can you find information about locating clients, fees, ethics, professional advertising, developing business plans, and writing contracts? Try reading *How to Become a Successful Consultant in Your Own Technical Field.*

Associated Technology Company, 804 Jordan Lane, Huntsville, AL 35816; $25.

Keep up with the changes affecting the outlook for digital services in general and TI services and products in particular by reading a study report called *G-114, The Outlook for TI Services and Equipment.* The study discusses the change to all-fiber transmission media and to network-oriented intelligent multiplexers, factors that forecast very high average annual growth rates through 1998.

Business Communications Co., 25 Van Zant Street, Norwalk, CT 06855; (203) 853-4266; $2,250.

A country-by-country listing of 1,000 academic, commercial, industrial, and government research entities in 42 countries appears in *The World Yearbook of New Generation Computing Research and Development.* The 600-page directory contains a section devoted to a survey of computer networks that provide researchers with an electronic communications infrastructure.

Ergosyst Associates, 910 Massachusetts Street, Lawrence, KS 66044; (913) 842-7334; $150 plus shipping and handling.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Interest Card.

Low 183 **Medium** 184 **High** 185

Mailbag

Continued from p. 3

"All of this issue is very interesting because of the TRON architecture. Congratulations." J.C.P., Cuautla, Mexico

"This issue on Japanese microprocessor developments was very enlightening." B.U.V., San Diego, CA

"I would like an in-depth article on the Texas Instruments DSP TMS320C30." F.A.G, Best, Netherlands (I am personally not familiar with this device, however, I will forward your request to the guest editors of the December special issue on DSP.—J.F.)

"...liked GaAs microprocessors and digital systems...more on advanced computer architecture and evaluations of bus standards." A.M., Tehran, Iran

"I like COMPMAIL+ and the dedicated bulletin board for use by the Computer Society's Working Groups...." J.A.G., Fishkill, NY

"Good magazine." D.H., Austin, TX (Thank you. Always good to hear from my old hometown.—J.F.)

" (Like to see) I. articles discussing multi-microprocessor systems; 2. memory architecture design trend for future microprocessors." Y-H.W, Taipei, Taiwan, China

"(More) 1. about expert systems for AI; 2. detail about public-domain microsystems." B.G.Y., Inchon, Korea

"I liked New Products." J.K., Froideville, Switzerland

"I liked MicroView...more on graphics hardware." Y.O., Memmartins, Portugal

"I liked the lot. I disliked that there wasn't more of it. It's a very good magazine...more on RISC/WISC, signal/image processing, OS, etc." S.G., Newcastle upon Tyne, UK

"I liked the whole magazine. It is an excellent publication...more tutorial-like articles...more in-depth reviews of new products." M.R., Astoria, NY

"I liked GaAs R&D efforts." D.P., Grand Forks, ND

"More on RECENT microprocessor designs." T.J-C., Taiwan, Taipei, China

"More on computer architectures, parallel processing, and MIMD machines." N.M., Campinas, Brazil

"I liked MicroReview....Keep up the good work. I disliked VideoRAM article. Cover teaser promised more. I felt it was difficult to read. (More on) 1. data communications, protocols, and microprocessors to support them; 2. how microprocessors will affect OS development. (Unix, etc.)" B.G.C., Irving, TX

"I liked "The IMS T800 Transputer"...MicroLaw, MicroReview. I disliked nil (nothing)...more ISDN and bus standards." A.V., Bombay, India

"All of the articles are good...more articles about design automation development." M.F., Bandung, Indonesia

"I liked the article's name: "A Fast Integer Binary Logarithm of Large Arguments," among others...more on signal processing and graphics handling on PC/AT, PS/2." T.T., Espoo, Finland

Letters to the Editor

Continued from p. 3

clude countries such as Egypt, South Africa, Thailand, and Turkey that Karjala mentions and whose significant contribution to copyright losses of billions of dollars he finds "implausible to the point of incredibility."

What I believe he is overlooking is that when there's piracy in these countries it's not merely domestic piracy. The reason for their significant contribution to the losses of US copyright proprietors is that the copying isn't done just for distribution *within* the country, but for *export* of the pirated copies to other countries throughout the world—including the United States. In other words, we can't, as Stern suggests, merely look at "the major software markets [all of which] seem to have adhered to both of these treaties...."

Copyright protection in these countries can be achieved through "backdoor," simultaneous publication in the US and in a Berne country, but that process is expensive. IBM spends approximately $10 million annually to publish simultaneously, and small US authors and proprietors usually have to forgo Berne protection because they can't afford the expense. Also, US reliance on the "backdoor" [approach] wins us no friends in the international copyright community (we've sought Berne benefits without adherence), and it's inconvenient and uncertain.

Peter F. Nolan, who testified before Congress for the Motion Picture Association of America, cited an instance in which senior officers of an American motion picture company had to make multiple trips to Thailand to give evidence on the issue of simultaneous publication; nevertheless the Thai court ultimately ruled against the company on that issue. Berne adherence would give US copyright proprietors protection in these 24 countries without the need for simultaneous publication.

Berne adherence will significantly increase the likelihood of achieving effective enforcement of intellectual property protection through the General Agreement on Tariffs and Trade (the GATT). Inclusion of an intellectual property code in the GATT would provide US copyright owners with a much-needed enforcement mechanism to safeguard their rights. However, US negotiators cannot credibly urge the inclusion of high Berne-level standards of copyright protection if the US is unwilling to join Berne.

Thus, Berne offers clear benefits to US copyright proprietors by providing them with a better opportunity to protect their rights abroad.

What is the "cost" of adhering to Berne? Because relatively few changes are required to make our copyright law Berne-compatible, the cost of adhering is minimal. Copyright notice will no longer be mandatory, but the implementing legislation contains incentives to encourage its continued use by copyright proprietors. The implementing legislation will leave our system of registration intact, save for a modification of the provision requiring an author or proprietor, if he or she has not already done so, to register before suit for copyright infringement. The important incentives to encourage registration will be retained. For example, a registration certificate will continue to be prima facie evidence of the copyright's validity if the registration is made within five years after publication, and statutory damages and attorney's fees can still be requested if the registration is made prior to the infringement or within three months after publication.

That the benefits of Berne significantly outweigh any costs of adherence is illustrated by the wide-ranging interests that support the implementing legislation. They include not only authors and copyright proprietors in various industries (for example, motion pictures, music, software, databases, publishing), but also user groups, including the American Library Association, the National School Boards Association, and the Association of Research Libraries.

Concerns raised by Stern and Karjala are unfounded. It is indisputable that Berne requires granting authors certain basic rights—including reproduction, adaptation, public performance, broadcasting, and translation rights—for a specified term. Berne does not, however, specify how the line must be drawn between unprotectible ideas and protectible expression in any given work or category of works. US courts will continue to determine the appropriate scope of copyright protection for computer programs, and which nonliteral elements of a program are protectible, on the basis of US law.

Both the House and Senate bills (and the committee reports that accompany them) state explicitly, and in exhaustive detail, that US obligations under Berne may be performed only pursuant to domestic law, and that no action can be brought pursuant to the provisions of the Berne Convention itself. Thus, adherence to Berne will not alter the scope or development of copyright protection for computer programs in the US.

Both Stern and Karjala expressed concern over the possibility of differential treatment of works of US and foreign authors. Stern said, "I would like to hear more details about a proposal that we should have a system in which US owners of software copyrights get lesser rights than foreign owners." That's not a proposal that I or any of the numerous other people supporting Berne adherence has made. Berne addresses only how a nation treats works of foreign origin; under Berne article 5(3), copyright protection for works of domestic origin is strictly a matter for that country's domestic law. Berne permits differential treatment; but it by no means requires it, and no one has really suggested it. The Berne legislation now in Congress makes no distinction as to the scope of copyright protection between US and foreign works.

The only differential treatment Congress currently contemplates is a minor one, as to section 411, registration as a prerequisite to suit. Almost the entire registration system in this country is permissive rather than mandatory. The permissive part will continue but Berne prohibits us from imposing the mandate of section 411 as a formality for foreigners.

A likely compromise between the Senate bill (which would remove section 411) and the House bill (which would leave it intact) is a requirement that would apply only to works of domestic origin. However, as a practical matter, such a compromise would place no burden on proprietors of US works that they don't have under the law as it exists now. And even now, it's not the requirement of section 411, but the significant incentives to registration elsewhere in the copyright law, that prompt the overwhelming bulk of the registrations. Only a few thousand infringement suits are filed each year, and most of them are on works long since registered for other reasons.

In sum, Berne offers significant, tangible benefits to US authors and copyright proprietors, at minimal cost. Berne will aid significantly in the fight against foreign piracy of US copyrighted works, and unsupported fears about the implications of adherence should not stand in the way of US adherence.

Gunter Hauptman
Purchase, New York

Department

New Products

Marlin H. Mickle,
University of Pittsburgh

Three RISCs at the ready

If not a byword, the RISC is an industry trend. New Products profiles three reduced instruction set computer offerings: Two are optimized for embedded-controller functions, while the third offers mainframe performance in a modular approach.

Intel's 32-bit, 80960 RISC architecture offers parallelism and modular features for embedded control applications—numerics processing, robotics, and high-speed telecommunications. According to the company, the single-chip 80960KB and 80990KA processors each perform at 7.5 (VAX) MIPS and 15,000 Dhrystones. The 80960KA is the same configuration as the 80960KB without an FPU.

On-chip 80960KB functions include 32 × 32-bit registers, a 512-byte instruction cache, a stack frame cache, and a 32-bit multiplexed burst bus. An interrupt controller with 256 vectors features 32 levels of interrupt priority.

The 80960KB integrates a 4 M-Whetstone, on-chip FPU with a 20-MHz clock rate. The IEEE-standard FPU operates on 32-, 64-, or 80-bit arithmetic functions.

Both processors have a 32-bit burst bus that can load 4 sequential words with every address access and fill the instruction cache with 4 words of data matched to exact cache-line length. **Intel; $390 each in 100s (80960KB); $174 each in 100s (80960KA, Oct. 1988).**

80960KB **Reader Service Number 10**
80960KA **Reader Service Number 11**

Advanced Micro Devices is shipping 16-, 20-, and 25-MHz versions of the 32-bit Am29000 RISC for embedded-controller, Unix-workstation, and multiprocessing-system applications. The 17-MIPS, 25-MHz version delivers 36,000 Dhrystones (V1.1 Dhrystone benchmark) with 8 Kbytes of instruction and data-cache memory. The Am29000 architecture also utilizes low-cost DRAM and video DRAM devices.

Features include 192 general-purpose register files, an on-chip, 64-entry MMU with single-cycle address translation, and a 512-byte, single-cycle branch-target cache.

The Am29000 uses a 1.2 micrometer CMOS wafer fabrication process and is available in 169-lead ceramic pin PGAs.

Hardware and software tools include optimizing compilers for C, a complete assembler, linker, loader, and a librarian incorporating simulation models. Optimizing compilers for Pascal, Fortran, and Ada are now under development.

Also forthcoming are three support chips—a floating-point processor, a data- transfer controller, and an integrated cache unit. Beta-site sampling of a 20-MIPS, 30-MHz Am29000 began in May. **Advanced Micro Devices; $174 each in 100s (16-MHz version); $320 each in 100s (20-MHz version); and $349 each in 100s (25-MHz version).**

16-MHz **Reader Service Number 12**
20-MHz **Reader Service Number 13**
25-MHz **Reader Service Number 14**

Motorola's 88000 RISC family includes a 14- to 17-MIPS 88100 microprocessor, an 88200 cache/memory management unit, and a series of multiprocessor boards called Hypermodules.

The 32-bit, 165,000-transistor 88100 integrates five execution units on chip: integer, floating-point adder, floating-point multiplier, instruction, and data. Floating-point functions achieve 7 to 12 Mflops. Units process concurrently while register scoreboarding enables 11 different operations at once. Special-function units allow AI or 3D graphics applications.

The 88200 CMMU is synchronized with 88200 clock rates and includes 16 Kbytes of cache memory storage and four-way, set-associative memory with zero wait states. Two 88200 chips supply the 88100 with instruction and data, respectively.

A preconfigured, 8 × 4-in. Hypermodule circuit board contains one 88100 and two 88200 CMMU chips. The company plans five versions topped off by a 50-MIPS array of four processors teamed with eight memory chips. **Motorola; $495 each in 100s (88100); $795 each in 100s (88200); $1,400 in 100s (Hypermodule: one 88100 and two 88200s).**

88100 **Reader Service Number 15**
88200 **Reader Service Number 16**
Hypermodule **Reader Service Number 17**

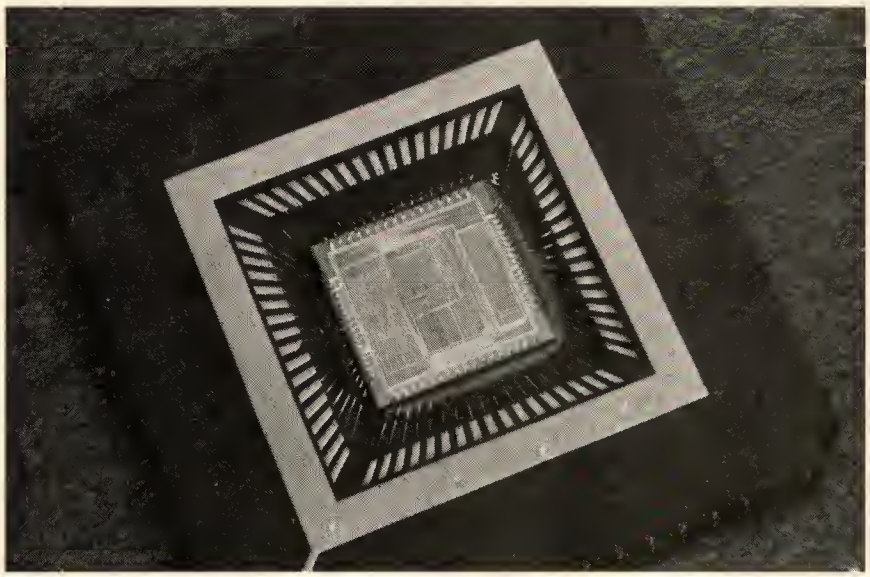
Multiplexer connects remote terminals

The Telebyte Model 570 accommodates eight RS-232, full-duplex data ports from the IBM 3174 and provides a two-twisted-pair, composite link.

The Model 570 Quick Mux remote terminal server package allows placement of eight ASCII terminals up to 5,000 feet away from an IBM 3174 Subsystem Control Unit. Connection of the 3174 to IBM's Token-Ring Network provides asynchronous ASCII devices a data path to the mainframe. Each input port can accept full-duplex data at any rate up to 19,200 bps. Each port also supports two bidirectional control signals. Power is supplied by a wall-mounted transformer through an 8-foot line cord. **Telebyte; $548 each with eight cables and modular adapters. OEM discount.**

Reader Service Number 18

NSC announces 16-bit microcontroller



An entire system and peripheral functions reside on a single-chip HPC Core megacell.

The CMOS HPC Core megacell contains a 16-bit architecture, six working registers, a microinstruction ROM, and four 16-bit timers/counters. Also featured are three 16-bit capture registers, eight interrupts, watchdog logic, a serial I/O interface, and one or more 256 × 8-bit RAMs. Memory, I/O, and peripheral registers are memory-mapped into a 64-Kbyte address space. The megacell performs a register instruction in 240 ns at 17-MHz clock frequency. Users can tailor the cell architecture to applications and migrate existing HPC designs to ASICs. **National Semiconductor.**

Reader Service Number 19

High-speed PC/AT controllers announced

A family of IBM PC/AT bus controllers supports one-to-one interleaving, an 8-Kbyte ring buffer, and a high-speed memory management system. The Omti 8140 and 8147 feature a half-card factor Winchester disk-drive support, while the 8247 provides combined Winchester and floppy disk-drive support in a full-length controller. The 8147 and 8247 incorporate run-length-limited 2,7 encoding with a 7.5-Mbit transfer rate to increase disk-drive capacity, and employ a 48-bit ECC polynomial error-detection and correction scheme. **Scientific Micro Systems; $120 (8140), $130 (8147), and $145 (8247) each in 100s.**

8140 **Reader Service Number 20**
8147 **Reader Service Number 21**
8247 **Reader Service Number 22**

Don't waste your time on logic

The single-chip Am29PL142 field-programmable controller doubles the memory of the Am29PL141 and features microinstruction capability. The 128-word × 34-bit PROM controller executes 29 high-level microinstructions—jumps, loops, and subroutine calls—that eliminate logic-equation programming. Intelligent control of system register files, ALUs, and I/Os is provided. The 20 MHz-controller contains nine conditional inputs and 16 externally expandable outputs. **Advanced Micro Devices. $24.95 each in 100s.**

Reader Service Number 23

Watchdog timer secures micros

The 80C521 and 80C321 CMOS microcontrollers feature watchdog timers programmable from 128 μs to 4 seconds at 12 MHz. Software- and pin-compatible with the 80C51, this pair also has dual data pointers to reduce external data-transfer bottlenecks. Software allows self-reset without external logic components. The 80C521 contains 256 bytes of RAM and 8 Kbytes of ROM; the 80C321 is without ROM. Emulators and C compilers serve as development tools. **Advanced Micro Devices; $8.50 each in 1,000s in plastic DIPs, plus $3,000 mask charge (80C521); and $8.85 each in 100s (80C321).**

80C521 **Reader Service Number 24**
80C321 **Reader Service Number 25**

Flash memories for embedded controllers

According to the company, the 256-Kbit 27F256/28F256 and the 64-Kbit 27F64 flash memories feature in-circuit, 1-second electrical erasure and 4-second reprogramming with EPROM-equivalent reliability. The 28-pin 27F256 has a multiplexed address and write-enable pin compatible with JEDEC-standard, 28-pin EPROMs. The 32-pin 28F256 features a nonmultiplexed, JEDEC-compatible pinout that accommodates density upgrades to 2 Mbits. The 27F64 is a direct socket replacement for standard 64-Kbit EPROMs.

All three CHMOS memories provide access speeds that eliminate wait states in most 8- to 16-bit microcontrollers and microprocessors. Military versions are planned for the first quarter of 1989. **Intel Corp.; $19.90 each in 10,000s (27F256/28F256), $8 each in 10,000s (27F64).**

27F256/28F256
Reader Service Number 26
27F64
Reader Service Number 27

High-speed companion saves power

The 1 M-bit Am27C010 UV EPROM delivers a 150-ns fast access time with a maximum standby current of 100 μA for power-sensitive applications. The CMOS 27C010 provides upgrades from 28-pin JEDEC EPROMs with 32-pin DIP and LCC packages. Five versions range from 150- to 300-ns access times. **Advanced Micro Devices; $29.05 to $54.95 each in 100s.**

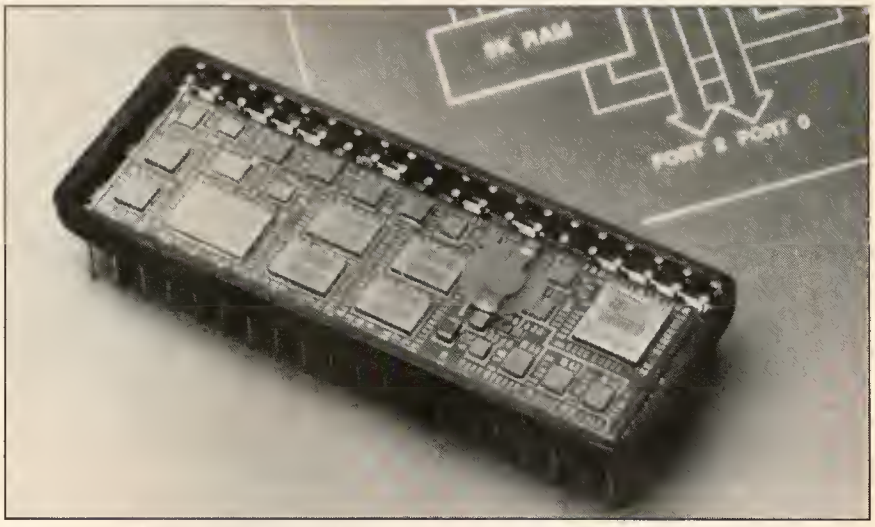
Reader Service Number 28

Supercomputer memory opens up

Systems designers can use the 213 VMEbus Host Interface to apply Dataram Wide Word 2000 memory on the VMEbus at 640-Mbyte capacity per Wide Word unit. The interface consists of a DMA controller board set, a general Wide Word interface, and two woven controlled-impedance cables. P2 connectors join the interface to the VMEbus backplane. The interface provides users with additional connectivity to VMEbus peripheral devices such as archival storage and array processors. **Dataram.**

Reader Service Number 29

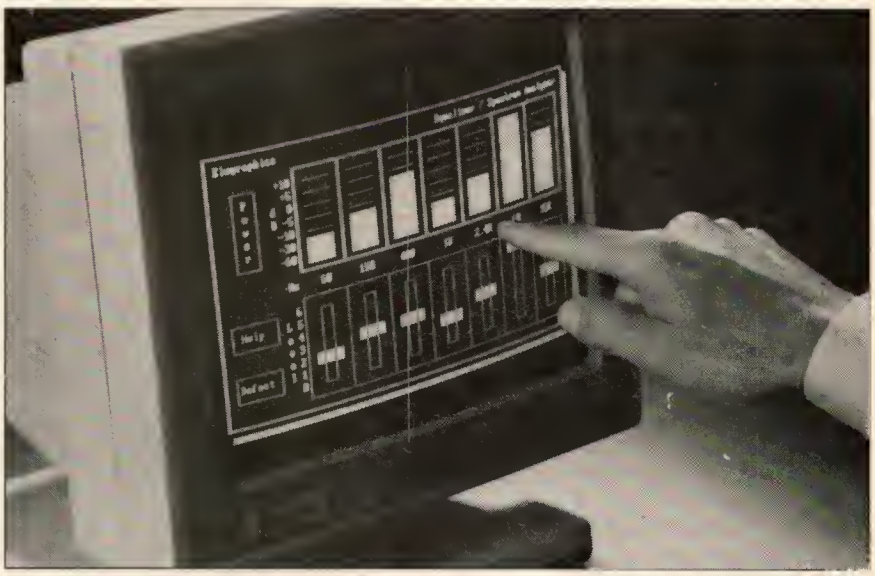
Hybrid accesses external memory



The Model C8-P31 Hybrid Microcontroller System is designed for "downhole," high-reliability, and military applications.

Developed around a CMOS 8031 microcontroller, the C8-P31 Hybrid Microcontroller System contains a 32K × 8-bit EEPROM, an 8K × 8-bit CMOS SRAM, and a built-in 14.74559-MHz crystal clock oscillator. Internal logic and decoding, a built-in monitor, and user-defined EEPROM for code or data memory are contained in a 40-pin DIP with standard 300-mil DIP centers. An open architecture provides external memory and peripherals for 128 Kbytes of accessible memory. A sleep mode feature consumes less that 1 mA. Operating between −55° C and + 150° C, the system is designed for hostile environments. **White Technology; $1,000 each in 100s.**

Reader Service Number 30



DuraTouch plastic-based touch screens are designed for use with EL panel, gas plasma, and liquid crystal displays, as well as flat and cyclindrically curved CRTs. Sizes range from 4 × 8-in. and 6 × 9-in. membranes for flat-panel displays to 9-, 11-, 13-, and 19-in. versions with standard CRT aspect ratios. Elographics.

Reader Service Number 31

The Magnavox 14-in. color VGA monitor offers analog input, variable vertical scanning frequencies up to 70 Hz, and a 31.5-MHz horizontal scanning frequency. Compatible with all VGA modes, the 9CMO82 monitor features a VGA interface cable that attaches to a 15-pin, high-density D connector. Philips Consumer Electronics; $649 (suggested retail).

Reader Service Number 32

SRAM attacks VMEbus bottleneck

The VMEsram board provides up to I Mbyte of high-speed nonvolatile SRAM with access times of 45, 60, or 110 ns. Access from Read Data Strobe can reach 10 ns. Compatible as a slave board to VMEbus Revision C, the VMEsram operates without the overhead incurred by error detection and correction and provides equal access/cycle times without refresh overhead. **Ovation Systems.**

Reader Service Number 33

Erase before you write

Three nonvolatile read/write EEPROMs in 1-, 2-, and 4-Kbyte configurations feature multiplexed address and data buses that directly interface with most microcontrollers and microprocessors. The 128 × 8-bit C10, the 256 × 8-bit C20, and the 512 × 8-bit C40 have a programming capability of 20,000 data changes per bit. The automatic erase-before-write memories come in 300-mil, 18-pin plastic DIP packaging. **National Semiconductor; $7.50, $9.00, and $10.50 each, respectively, in 100s.**

C10 Reader Service Number 35
C20 Reader Service Number 36
C40 Reader Service Number 37

Fast SRAMs leave tiny print

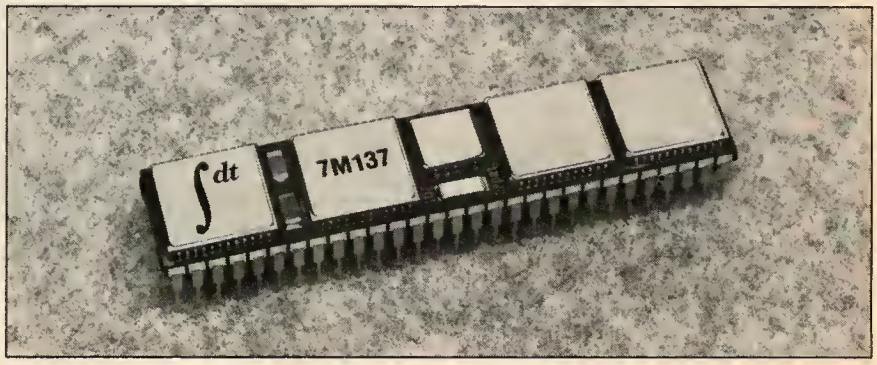
Two 64-Kbit CMOS fast SRAMs are available in small outline J lead packaging that allows a large plastic body with a minimum footprint and protects leads during handling. The 8 K × 8-bit, 35- to 45-ns MCM6264J matches DSP applications, while the 16K × 4-bit MCM6290J has a 12- or 15-ns output-enable function for fast data-cache access. The MCM6290J reaches 25-, 30-, and 35-ns access times. **Motorola; $13.69 to $22.13 each in 100s (MCM6264J), and $15.46 to $25.12 each in 100s (MCM6290J).**

MCM6264J Reader Service Number 38
MCM6290J Reader Service Number 39

Dual-port SRAMs eliminate wait states

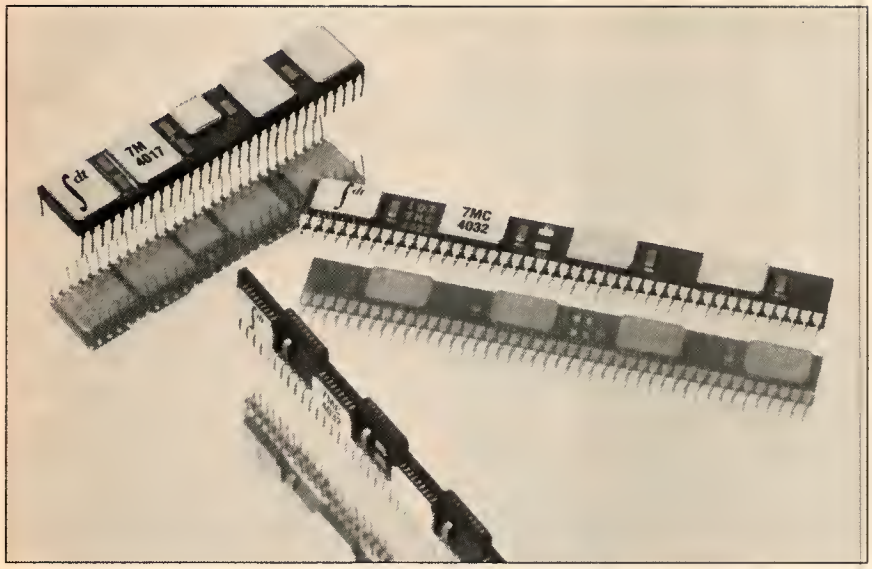
According to the company, the IDT7MI37 dual-port SRAM module replaces I0 monolithic chips in one-fifth the board space. Two ports with separate control, address, and I/O pins permit simultaneous asynchronous access to the total address space at full speed without hardware arbitration. Eight 4K × 8-bit, dual-port RAMs in LCC packaging and two decoder units comprise the 256-Kbit module. Access times range from 55 to 90 ns. **Integrated Device Technology; $259 to $507 each in 100s.**

Reader Service Number 34



The IDT7M137 provides access to shared memory for two or more processors with no wait states.

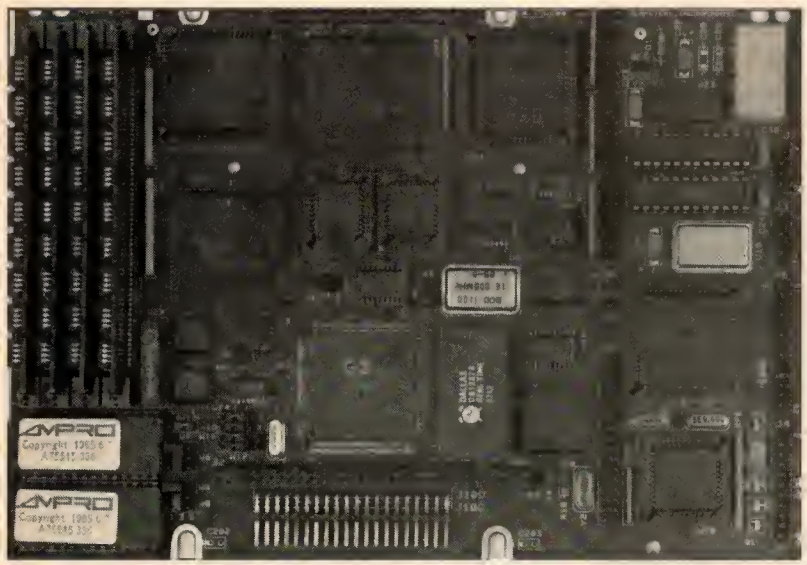
Multimegabit duo



CMOS IDT7M4017 and IDT7M4032 modules offer 30- to 70-ns access times.

IDT offers two SRAM modules with 32-bit-wide memories. The 64 K × 32-bit IDT7M4017 contains eight 32 K × 8 SRAMs in LCC packages mounted on a 60-pin, 600-mil-wide Cerdip substrate with onboard decoders. The IDT7M4032, featuring an 88-pin dual Cersip, uses eight 16 K × 4-bit SRAMs. Three 520-mil-high modules stack in a 4.5-sq.-in. board space. **Integrated Device Technology; $620 to $1, 240 each in 100s (IDT7M4017); $198 to $310 each in 100s (IDT7M4032).**

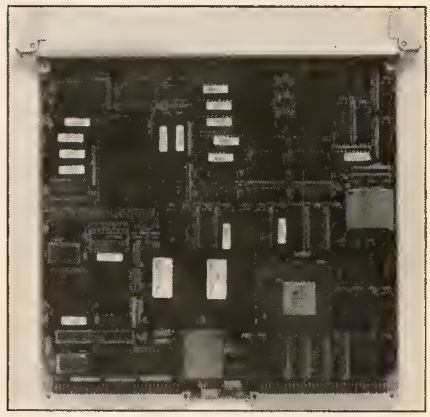
IDT7M4017 **Reader Service Number 40**
IDT7M4032 **Reader Service Number 41**



According to the company, the 46-cu.-in. Ampro Little Board/286 for embedded applications and harsh environments includes functions equal to those of an IBM PC/AT motherboard and three expansion cards. The PC/AT-compatible system offers 1 Mbyte of RAM and a set of peripheral controllers in an 8 × 5¾ × 1-in. package. Ampro; $739 (small OEM quantities).

Reader Service Number 42

Controller speeds disk caching



The SCSI/ANSI- and SASI-compatible CD22/4500-2048 Controller Board offloads data storage and manipulation tasks from the host CPU.

The SCSI Controller Board for Multibus II systems features high-speed disk caching and command sorting with a 12.5-MHz 80186 CPU and 2 Mbytes of parity-protected RAM. An SCSI interface communicates with up to seven other host/peripheral adapters. An 82258 Advanced DMA Controller transfers data at 1.5 Mbytes/s (asynchronous) and 4 Mbytes/s (synchronous). A local floppy disk interface provides low-cost data storage and supports both standard and higher capacity floppy drives. **Central Data; $3,055 each in 100s.**

Reader Service Number 43

CMOS devices meet SCSI standard

The DP8490 and DP5380 asynchronous SCSI devices feature acknowledge-to-request times of 60 ns and request-to-acknowledge times of 30 ns with a 30-ns read-across time. The DP8490 permits a DMA rate of 4 Mbytes/s. The DP5380 eliminates software changes necessary for the DP8490. Both devices are pin-compatible with NMOS NCR5380 SCSI devices, comply with the SCSI standard, consume a maximum of 10 milliamps, and come in 40-pin DIPs or 44-pin PLCCs. **National Semiconductor; $8 each (DP8490), and $7.50 each (DP5380).**

DP8490 **Reader Service Number 44**
DP5380 **Reader Service Number 45**

Flexible disk expands your storage

The "crashless" HyperFlex disk drive combines Winchester and standard floppy-disk drive technologies for IBM PC, PC/AT, AT, PS/2 Model 30, PC-compatible, and Apple computers.

The HyperFlex disk drive for OEMs offers a 20-Mbyte (formatted) and 24-Mbyte (unformatted) storage capacity on a single flexible disk. The 5.25-in., half-height drive can be internally mounted or function as an external subsystem. Protected storage disks do not require sensitive handling. The driver's embedded SCSI controller provides transparent defect mapping and error correction with data-transfer rates of up to 11 times the speed of a 360-Kbyte floppy drive. **Data Technology; less than $600 each in OEM quantities.**

Reader Service Number 46

The 2020 series of hand-held radio frequency terminals provides data communications in industrial materials-handling applications. Model 2020-IS performs inventory control in flammable atmospheres, and the 2020-LT operates in temperatures as low as -22° F. LXE.

Reader Service Number 47

SCSI chips support split-bus architectures

According to the company, four chips based on NCR's 53C90 SCSI controller achieve standard transfer rates at 2 percent of the usual bus-overhead time. NCR 53C90A and 53C90B hardware supports commonly used SCSI-2 bus sequences. The 53C90B also increases data integrity by means of a bit for parity passing.

The split-bus, 16-bit 53C94 and 53C95 chips allow CPU command-queuing simultaneously with DMA access and use an 18-bit DMA channel for reduction of memory bus accesses. The 53C95 adds control lines for differential-pair transceivers. The 53C94 and 53C95 can be programmed for either shared or split buses and are software-compatible with the 53C90 to minimize firmware changes. **NCR; $20.44 each in 1,000s (53C90A and 53C90B), $25.55 each in 1,000s (53C94 and 53C95).**

53C90A **Reader Service Number 48**
53C90B **Reader Service Number 49**
53C94 **Reader Service Number 50**
53C95 **Reader Service Number 51**

AppleCrate, Seagate connect

The AppleCrate SCSI platform provides resellers with cabinet, internal and external cables, software, and documentation. Resellers add a 20 to 80-Mbyte, half-height, 5.25-in. Seagate hard-disk drive for the Mac Plus, Mac SE, or Mac II. An additional SCSI port allows the daisy-chaining of up to seven SCSI devices. Supplied software includes utilities for formatting, initializing, testing, and verification of the disk. **AppleCrate.**

Reader Service Number 52

Develop and test real-time systems

The Link/X Development System consists of a dedicated Heurikon processor board compatible with Multibus I and II and VMEbus. The system communicates through high-speed bus ports with up to 16 processors that run a Vrtx real-time executive program. Heurikon's BusLink software supports C language source-level interactive debugging of Vrtx programs through the Unix system. **Heurikon.**

Reader Service Number 53

Lab computer features 24 expansion slots

The Scientific Laboratory System 5550 provides researchers with a 1-million-samples-per-second data acquisition and analysis system, 33,000 samples-per-second throughput to disk, and nine slots for applications requiring multiple data-acquisition modules. Fifteen system bus slots accommodate vector and floating-point accelerators, tape and disk systems, and graphics devices.

The system includes a 142-Mbyte Winchester disk and an internal graphics processor with a four-plane 640 × 480 resolution. Using a Motorola MC68020 CPU and an MC68881 math coprocessor, the SLS 5550 provides 3 MIPS of computation power and exceptional floating-point performance. Triple-bus architecture permits simultaneous, high-performance data acquisition and real-time graphics display. **Masscomp; $27,500 (hardware and software).**

Reader Service Number 54

The SLS 5550 runs third-party programs on the real-time Unix operating system and includes software for designing experiments without traditional programming.

Designed for the 80386

The multitasking kernel VRTX32/386, the advanced debugger RTscope 386, and the VRTX32 system monitor comprise a comprehensive real-time operating system for 80386-based applications. The VRTX32/386 executive isolates tasks by using local descriptors and runs tasks at different privilege levels to protect system data structures. VRTX32/386 tasks can operate in virtual 86 mode. Maximum interrupt-latency time is 14.5 μs for a 16-MHz 80386 processor with no wait states.

The RTscope 386 specialized debugger monitors VRTX32's system objects while operating concurrently with the multitasking system. It can display the contents of local and global descriptor tables and can set and remove hardware breakpoints.

The VRTX32 is also available for the Motorola 68000 family, the AMD Am29000, and the National Series 32000. **Ready Systems; $6,775 (VRTX32/386), $3,000 (RTscope 386).**

VRTX32/386 **Reader Service Number 55**
RTscope 386 **Reader Service Number 56**

Productivity tools aid system developers

The Integration Toolkit builds, tests, and debugs board-level and real-time uniprocessor and multiprocessor systems. It includes a system configuration generator, a real-time multiprocessor message-passing operating system, a range of device servers, a host simulator, a real-time multiprocessor system-level debugger, and support tools. The toolkit encourages the reuse of proven software components and works in a host/target environment. It runs on Unix workstation and terminal-based systems, VAX VMS systems, or PC/MS DOS-compatible hosts. **Multiprocessor Toolsmiths.**

Reader Service Number 57

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Interest Card.

Low 180 **Medium** 181 **High** 182

Product Summary

Marlin H. Mickle
University of Pittsburgh

For more information, circle the appropriate Reader Service Number on the Reader Service Card at the back of this magazine.

MANUFACTURER	MODEL	COMMENTS	R.S. NO.
Chips			
Advanced Micro Devices	Am29C327 processor	Double-precision CMOS, single-chip, 64-bit floating-point processor comes in 100-ns and 120-ns versions. Supports IEEE-754; DEC D, F, and G; and IBM formats. Performs in either flowthrough or pipelined modes. $595 (100 ns); $395 (120 ns) in 100-piece quantities; 169-lead PGA packages.	80
NCR Microelectronics Division	20C12 modem	Digital modem supports the Highway Addressable Remote Transducer protocol for communications between control room equipment and remote field devices. Also supports Bell 202 modem standards. $9.50 (sample pricing) in quantities of 1,000.	81
Boards			
Arcom Control Systems	SC280 module	Evaluation and system design module interfaces Zilog's 10-MHz Z280 processor to the asynchronous STEbus. Integrates the on-chip cache, pipeline, MMU, DMA, and serial I/O features of the Z280 with up to 96 Kbytes of static RAM, 128 Kbytes of EPROM, two additional serial channels, and the STEbus interface. £295.	82
Burr-Brown/ AT&T	DSPeed processor	Based on the 32-bit, floating-point WE DSP32 digital signal processor, this PC-compatible board features a 64-Kbyte RAM, a serial interface, a parallel interface to the PC bus, a utilities debugging program, and a utilities library. Software support tools include an assembler/linker, a simulator, and a C compiler operating under an MS-DOS operating system. $995 (quantity orders); OEM discounts available.	83
Comark Corporation	CAT 386 processor	Eight-layer, IBM AT-compatible 386 processor is based on Intel's 80386 CPU and Chips and Technology's chip set. Features Award BIOS, 16-MHz 80386, 32-bit data path, modular memory expansion, 16-Mbyte RAM capacity, 80387 coprocessor socket, test port, 16/8-bit I/O data path, 24-bit address bus, and AT-passive bus architecture. Operates with most IBM-compatible expansion cards and software. Measures 1.0 × 5.5 × 13.3 inches. $1,595 (16-MHz version); RAM pricing varies.	84

MANUFACTURER	MODEL	COMMENTS	R.S. NO.
DSP Design	SV25 processor	CMOS STEbus processor based on NEC's V25 8086 microcontroller generates target code with MS-DOS software development tools. Runs at 8 MHz. Offers two RS-232 serial ports, 24 digital I/O lines, two 16-bit programmable counter/timers, a dual-channel DMA controller, a programmable interrupt controller, and a watchdog circuit. No price given.	85
Heurikon	HK68/V20 processor	Based on Motorola's 68020 microprocessor, the VME-compatible computer boosts Unix and real-time systems. Offers a 32-bit data path, 32-bit addressing, a choice of CPU clock rates up to 25 MHz, and up to 4 Mbytes of dual access memory with two parity bits per word and hardware refresh. Options include paged MMU and 68881 floating-point coprocessor. No price given.	86
Logos Computer	LGII-286 motherboard	Designed with surface mount technology, the 16-MHz 80286 board connects with IBM PC AT and OS/2 computers. The 8-1/2 × 12-inch board contains two 8-bit and six 16-bit expansion slots and 80287 math coprocessor sockets. One megabyte of RAM can be installed on board. $495.	87
Software			
Intel Corp.	iRMX Source control system	Software system manages the development of iRMX operating system applications on iRMX systems. Controls access to source files, tracks changes to those files, and supports PL/M, Assembler, Fortran, C, and Pascal. $2,000 (single-system license).	88
	iRMX II.3 operating system	Release 3 of the real-time operating system provides support for the Multibus II architecture. Offers message passing via the MPC, the use of interconnect space, software configuration of boards, and centralized time-of-day clocking. Buffer pools increase the performance of applications that require quick memory recycling. $5,500 (development license for one CPU).	89
	Multibus I X.25	Communications protocols interconnect Multibus I systems to X.25 packet-switch communications networks. The design allows host computer access as a data terminal equipment device and as a data circuit-terminating device. Includes an application interface library and an interactive utility package. $1,300 each.	90
Miscellaneous			
Intel Corp.	Bitbus Starter Kit	Kit allows first-time Bitbus network users to construct and operate a simple network. Runs on an IBM PC or compatible host and contains analog and digital boards, a power supply, cables, and a PC gateway to the network. $3,350 each.	91
	iSBC 322/324 memories	User-installable modules upgrade iSBC 286/1X 80286-based Multibus I processor boards with either 2 or 4 Mbytes of zero-wait-state DRAM. $1,750 (iSBC 322); $3,250 (iSBC 324).	92

Advertiser/Product Index

FOR DISPLAY ADVERTISING INFORMATION, CONTACT:

Northern California and Pacific Northwest: Roy McDonald Assoc. Inc., 5915 Hollis St., Emeryville, CA 94608; (415) 653-2122.
Jim Olsen, P.O. Box 696, Hillsboro, OR 97123; (503) 640-2011.
Southern California and Mountain States: Richard C. Faust Co., 24050 Madison St., Suite 100, Torrance, CA 90505; (213) 373-9604.
Southwest: The House Co., 3000 Weslayan, Suite 345 Houston, TX 77027; (713) 622-2868.
Midwest: Robert Acker & Assoc., 480 Central Ave., Northfield, IL 60093; (312) 441-6050.
East Coast: Atlantic Representative Group, 349 Maple Place, Keyport, NJ 07735; (201) 739-1444.
New England: Arpin Associates, PO Box 227, Weston, MA 02193; (617) 899-5613.
Europe: Heinz J. Gorgens, Parkstrasse 8a, D-4054 Nettetal 1 - Hinsbeck (F.R.G.); phone: (0 21 53) 8 99 88; telex 841(17)2153310=HJG tlx d.

Advertising Director: Dawn Peck. For production information, conference, and classified advertising, contact Heidi Rex or Marian Tibayan.

IEEE MICRO, 10662 Los Vaqueros Cir., Los Alamitos, CA 90720; phone (714) 821-8380; Fax (714) 821-4010.

READER SERVICE CARD

IEEE MICRO

August 1988 issue
(card void after February 1989)

Name_____

Title_____

Company_____

Address_____

City_____ State___ ZIP_____

Country_____ Phone (___) _____

Please send
(Circle those you want):

201 Publications catalog
202 Membership information
203 Student membership information
204 IEEE senior member application
205 *IEEE Micro* subscription information

Reader interest
(Circle what you liked, add comments on the back)

Readers,
Indicate your interest in articles and departments **by circling the appropriate number** (shown on the last page of articles/departments) **in the shaded section of this card under Product Inquiries.**

Product inquiries **1**
(circle the numbers for products and advertisers you want more information on)

1	21	41	61	81	101	121	141	161	181
2	22	42	62	82	102	122	142	162	182
3	23	43	63	83	103	123	143	163	183
4	24	44	64	84	104	124	144	164	184
5	25	45	65	85	105	125	145	165	185
6	26	46	66	86	106	126	146	166	186
7	27	47	67	87	107	127	147	167	187
8	28	48	68	88	108	128	148	168	188
9	29	49	69	89	109	129	149	169	189
10	30	50	70	90	110	130	150	170	190
11	31	51	71	91	111	131	151	171	191
12	32	52	72	92	112	132	152	172	192
13	33	53	73	93	113	133	153	173	193
14	34	54	74	94	114	134	154	174	194
15	35	55	75	95	115	135	155	175	195
16	36	56	76	96	116	136	156	176	196
17	37	57	77	97	117	137	157	177	197
18	38	58	78	98	118	138	158	178	198
19	39	59	79	99	119	139	159	179	199
20	40	60	80	100	120	140	160	180	200

READER SERVICE CARD

IEEE MICRO

August 1988 issue
(card void after February 1989)

Name_____

Title_____

Company_____

Address_____

City_____ State___ ZIP_____

Country_____ Phone (___) _____

Please send
(Circle those you want):

201 Publications catalog
202 Membership information
203 Student membership information
204 IEEE senior member application
205 *IEEE Micro* subscription information

Reader interest
(Circle what you liked, add comments on the back)

Readers,
Indicate your interest in articles and departments **by circling the appropriate number** (shown on the last page of articles/departments) **in the shaded section of this card under Product Inquiries.**

Product inquiries **2**
(circle the numbers for products and advertisers you want more information on)

1	21	41	61	81	101	121	141	161	181
2	22	42	62	82	102	122	142	162	182
3	23	43	63	83	103	123	143	163	183
4	24	44	64	84	104	124	144	164	184
5	25	45	65	85	105	125	145	165	185
6	26	46	66	86	106	126	146	166	186
7	27	47	67	87	107	127	147	167	187
8	28	48	68	88	108	128	148	168	188
9	29	49	69	89	109	129	149	169	189
10	30	50	70	90	110	130	150	170	190
11	31	51	71	91	111	131	151	171	191
12	32	52	72	92	112	132	152	172	192
13	33	53	73	93	113	133	153	173	193
14	34	54	74	94	114	134	154	174	194
15	35	55	75	95	115	135	155	175	195
16	36	56	76	96	116	136	156	176	196
17	37	57	77	97	117	137	157	177	197
18	38	58	78	98	118	138	158	178	198
19	39	59	79	99	119	139	159	179	199
20	40	60	80	100	120	140	160	180	200

SUBSCRIPTION CARD

SUBSCRIBE TO IEEE MICRO

☐ YES, sign me up! ☐ Renew my subscription!

If you are a member of the Computer Society or any other IEEE society, pay the member rate of only **$18** for a year's subscription (6 issues). Subscriptions are annualized with membership. For orders submitted March through August, please pay half the given full-year fee for a half-year's subscription.

Society:_____ IEEE membership no:_____

☐ YES, sign me up! ☐ Renew my subscription!

If you are a member of ACM, NSPE, ACS, IEE (UK), SCS, IPSJ, IECEJ, or any other technical society, pay the sister society rate of only $27 for a year's subscription (6 issues).

Society:_____ Mem. no. (if any):_____

_____
Full Signature *Date*

_____
Name

_____
Street

_____
City

_____
State/Country *ZIP/Postal Code*

☐ Payment enclosed
☐ Charge to ☐ Visa ☐ MasterCard ☐ AmEx

☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐
Charge Card Number

Mo. Yr.
☐☐ ☐☐
Exp. Date

M888

Charge orders also taken by phone:
(714) 821-8380 8:00 a.m. to 5:00 p.m. Pacific time
Circulation Dept.
10662 Los Vaqueros Cir.
Los Alamitos, CA 90720

Editorial comments

I liked: _____

_____

_____

_____

_____

I disliked: _____

_____

_____

I would like to see: _____

_____

_____

_____

_____

For reader service inquiries, use other side

PLACE STAMP HERE

PO box address for reader service cards only

IEEE MICRO

Reader Service Inquiries
PO Box 16508
North Hollywood, CA 91615-6508
USA

Editorial comments

I liked: _____

_____

_____

_____

_____

I disliked: _____

_____

_____

I would like to see: _____

_____

_____

_____

_____

For reader service inquiries, use other side

PLACE STAMP HERE

PO box address for reader service cards only

IEEE MICRO

Reader Service Inquiries
PO Box 16508
North Hollywood, CA 91615-6508
USA

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 38 LOS ALAMITOS, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Computer Society of the IEEE
Circulation Dept.
10662 Los Vaqueros Cir.
Los Alamitos, CA 90720-9970

THE COMPUTER SOCIETY

A member society of the Institute of Electrical and Electronics Engineers, Inc.

Use the Reader Service Card to obtain the following information:

- Membership application—student #203, others #202
- Periodicals subscription form for individuals #200
- Periodicals subscription form for organizations #199
- Publications catalog #201
- Standards working groups list #195
- COMPMAIL+ international electronic mail/database brochure #194
- Technical committee list/application #197
- Chapters lists, start-up procedures—student/regular #193
- Student scholarship information #192
- Awards description/nomination forms #198
- Volunteer leaders/staff directory #196
- IEEE senior member application #204

Purpose

The Computer Society advances the theory and practice of computer science and engineering, promotes the exchange of technical information among 90,000 members worldwide, and provides a wide range of services to members and nonmembers.

Membership

Members receive the acclaimed monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others seriously interested in the computer field.

Publications and Activities

Computer. An authoritative, easy-to-read magazine containing tutorial and in-depth articles on topics across the computer field; plus news, conferences, calendar, interviews, new products, etc.

Periodicals. The society publishes six magazines and three research transactions. Refer to membership application or request information as noted above.

Conference Proceedings, Tutorial Texts, Standards Documents. The Computer Society Press publishes more than 100 titles every year.

Standards Working Groups. Over 100 of these groups produce IEEE standards used throughout the industrial world.

Technical Committees. Over 30 TCs provide newsletters, interaction with peers in specialty areas, and have direct influence on standards, conferences, education, etc.

Conferences/Education. The society holds about 100 conferences each year and sponsors many educational activities, including computing sciences accreditation.

Chapters. Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

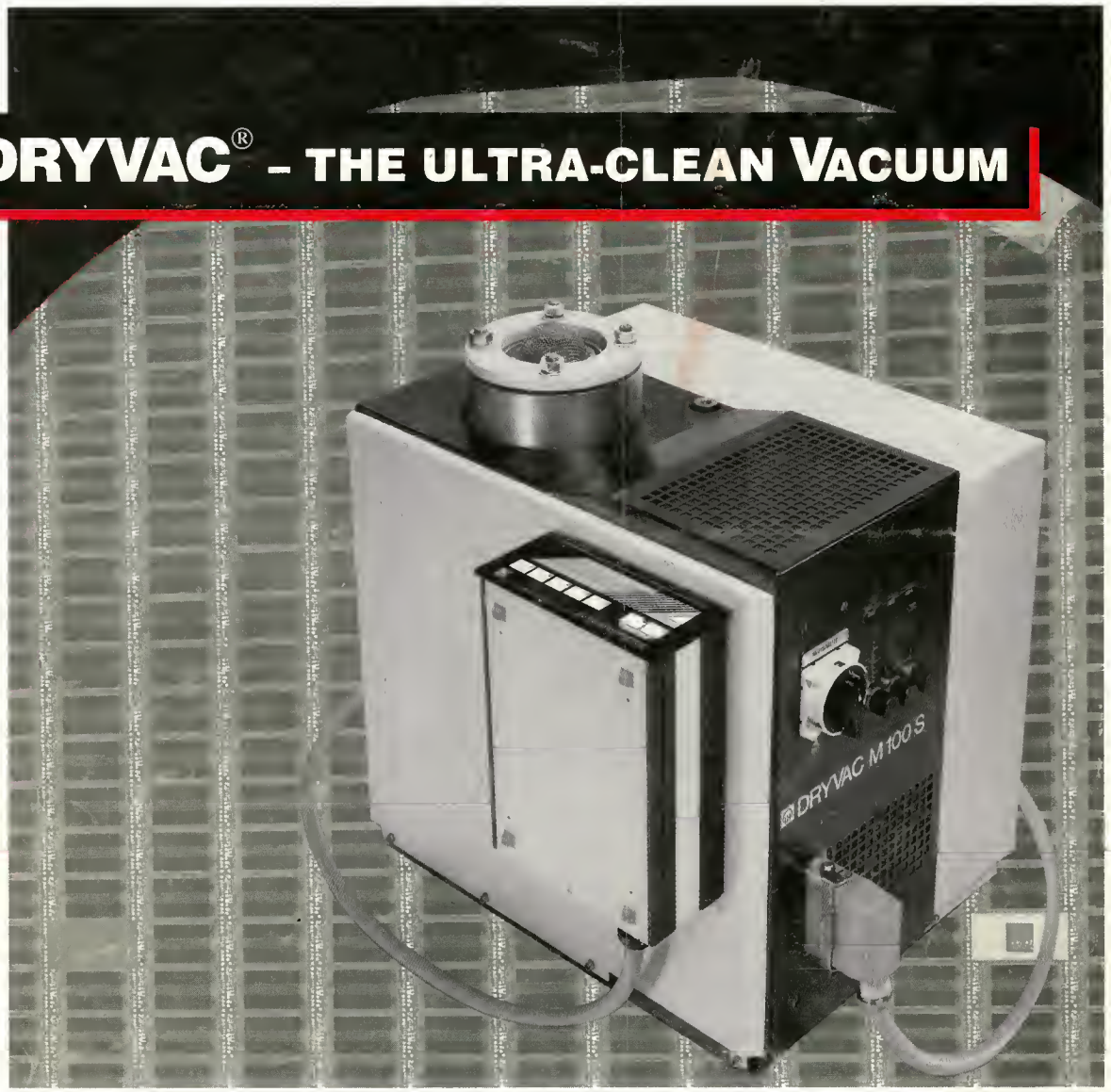
European Office

Payments for Computer Society membership and publication orders are accepted by cheques in Belgian, British, German, Swiss, or U.S. currency, or by American Express, Eurocard, MasterCard, or Visa credit cards.

Ombudsman

Members experiencing problems—magazine delivery, membership status, unresolved complaints—may write to the ombudsman at the Publications Office.

DRYVAC® – THE ULTRA-CLEAN VACUUM



171.8.61.12.004.02 CD

The slightest impurity in semiconductor production is bad news. But there is an answer to the problem of achieving absolutely clean manufacture: the dry compression DRYVAC has a completely oil-free vacuum.

Except when running at full power you would scarcely know that it was there, requiring very little space and fitting into any plant. The DRYVAC is economical with valuable inert gas thus reducing operating costs; and the lengthy maintenance intervals raise productivity.

The DRYVAC is operated and vital parameters in the processes are monitored by a control unit, fitted either directly to the pump or connected via a cable to a remote switchroom.

DRYVAC is the perfect technical solution for processes where only the purest vacuum is good enough.

LEYBOLD

Questions? Only too glad to help.
Call our nearest branch or the factory in Cologne.

Reader Service Number 3

LEYBOLD
VACUUM PRODUCTS INC.
5700 Mellon Road
Export, PA 15632

LEYBOLD AG
Bonner Strasse 498
D-5000 Cologne 51